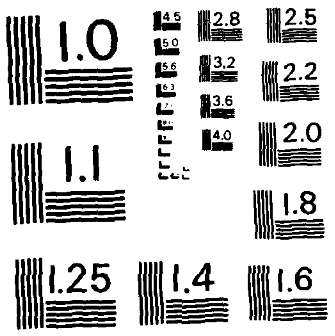
AD-A159 513 PROCEEDINGS OF THE COURSE ON ALGORITHMS AND DATA STRUCTURES FOR GEOMETRIC..(U) INSTITUT FUER INFORMATIK ZURICH (SWITZERLAND) G HEISER ET AL. 26 JUL 85 1/2

UNCLASSIFIED R/D-5134-CC/MA-02 F/G 12/1 NL


1.0
4.5
5.0
5.6
6.3
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.1
1.8
1.25
1.4
1.6

R+D-5134-CC/MA-02

2

Proceedings of the Course on Algorithms and Data Structures for Geometric Computations held at CISM in Udine, Italy, July 8-12 1985

G. Heiser, K. Hinrichs, A. Meier, J. Nievergelt
Institut für Informatik ETH Zürich
July 26. 1985

sponsored in part by ERO under grant DAJA 45-85-M-0318

(-A-)

The Outline of the course was as follows:

D. Stanat presented an introduction to algorithms and data structures of general interest.

1) *T. Ottmann* introduced the field of computational geometry; with a talk on geometric algorithms in the plane. He demonstrated geometrical divide-and-conquer on the problem of reporting intersecting linesegments. Then he discussed sweep line algorithms for the same problems, introducing the use of skeleton structures and exploring minimization of memory space requirements. The usage of plane sweep to report intersecting iso-oriented ~~rectangles led~~ to the introduction of segment trees, interval trees and priority search trees. ~~Zigzag-decomposition~~ of a graph was demonstrated on the problem of polygon intersection. Finally, a hidden line detection algorithm was presented.

2) *J. Hopcroft* discussed geometrical problems related to robotics; After giving an overview of the field, his talks centered on the two problems of automated generation of blending surfaces and on motion planning.

3) *P. Widmayer* discussed heuristics for finding approximations for Steiner minimum trees; The exact solution of this problem is known to be np-complete.

4) *K. Hinrichs* described the grid file as a data structure suited for geometrical computation and presented experience with the implementation.

5) *A. Meier* explained different schemes for representing three-dimensional objects; Furthermore he discussed how the Relational Data Base model could be used to store geometric objects. He pointed out some of the problems of this approach and indicated possible extensions of the relational model to make it suitable for computational geometry.

6) *F. Luccio* discussed visibility problems that occur in VLSI design; He showed the equivalence of a visibility problem with a planar graph and hence the applicability of graph theory. He also briefly explained the view of a program as a decision tree.

7) *F. Preparata* in his overview of geometric algorithms introduced algorithms for point location, convex hull and the maxima of a set of vectors in two and three dimensions; The time complexity of these algorithms was discussed. Algorithms for constructing Voronoi diagrams were introduced to solve proximity problems and the use of plane sweep algorithms for solving planar intersection problems was shown.

and 8) Additional contributions came from *J. Sack* and *Th. Strothotte* who presented a recent result on merging heaps and also pointed out some unsolved problems.

Further information on the talks can be found in the enclosed papers handed out by *J. Hopcroft*, *K. Hinrichs*, *A. Meier* and *F. Luccio*.

Zurich, July 26, 1985

Prof. J. Nievergelt

Rate on file

A-1

Centre International des Sciences Mécaniques — International Centre for Mechanical Sciences

I-33100 UDINE (Italy), Palazzo del Torso, Piazza Garibaldi, 18
Tel. (0432) 29 49 89 - 2 25 23

Course on

ALGORITHMS AND DATA STRUCTURES FOR GEOMETRIC COMPUTATION

Udine, July 8-12, 1985

TIME TABLE

Monday:	9. 00 - 10. 30	Registration
	10. 30 - 12. 30	D. Stanat: Introduction to Algorithms and Data Structures, I;
	14. 30 - 16. 30	D. Stanat: Introduction to Algorithms and Data Structures, II;
Tuesday:	9. 00 - 11. 00	T. Ottmann: Geometric Algorithms in the Plane, I;
	11. 30 - 12. 30	F. Luccio: Visibility Problems, I;
	14. 30 - 16. 30	J. Hopcroft: Robotics Algorithms, I;
Wednesday:	9. 00 - 11. 00	T. Ottmann: Geometric Algorithms in the Plane, II;
	11. 30 - 12. 30	K. Hinrichs: Data Structures for Geometric Computation;
	14. 30 - 16. 30	J. Hopcroft: Robotics Algorithms, II;
	20⁰⁰	Dinner: Hotel Astoria
Thursday:	9. 00 - 11. 00	A. Meier: Data Bases for Geometric Objects;
	11. 30 - 12. 30	F. Luccio: Visibility Problems, II;
	14. 30 - 16. 30	F. Preparata: Overview on Geometric Algorithms, I;
Friday:	9. 00 - 11. 00	F. Preparata: Overview on Geometric Algorithms, II;
	11. 30 - 12. 30	Discussions.

Possible contributed talks will be delivered at the end of the afternoon sessions.

Centre International des Sciences Mécaniques — International Centre for Mechanical Sciences

I-33100 UDINE (Italy), Palazzo del Torso, Piazza Garibaldi, 18
Tel. (0432) 29 49 89 - 2 25 23

List of Participants in the course

Algorithms and Data Structures for Geometric Computation

July 8-12, 1985

ARNOLDI Massimo, Computer Science Student
Via alla Roggia, 9a
6962 Viganello
Switzerland

CHAABAN Moustafa, Professor
Ain Shams University
Faculty of Engineering
15, El Farik El Masri st.
Almaza, Heliopolis, Cairo
Egypt

DELLA RICCIA Giacomo, Professor
Universita' di Udine
Ist. di Matematica, Informatica
e Sistemistica
Via Mantica, 3
33100 Udine
Italy

GAMBOSI Giorgio, Researcher
Istituto di Analisi dei Sistemi
ed Informatica - CNR
Viale A. Manzoni, 30
00185 Roma
Italy

GUMRUKCU Haluk,
Dept. Computer Engineering
Middle East Technical University
Ankara
Turkey

HEISER Gernot, Assistant
ETH-Zentrum
Institut für Informatik
CH-8092 Zürich
Switzerland

KLEIN Rolf, Research Assistant
Kollegium am Schloss, Bau IV
Institut für Angewandte Informatik
75 Karlsruhe
Western Germany

KOHLER Walter, Assistant
Chief of Techn. Data Processing
Tauernkraftwerke AG.,
Rainerstr. 29,
Postfach 161
A-5021 Salzburg
Austria

KOWALCZYK Bogdan, Specialist
Warszawa Inst. of Mechanical Engineering
ul. Narbutta, 86
Warszawa
Poland

LALEMENT Rene',Professor
ENPC - CERMA
B.P. 105
93194 Noisy Legrand
France

LAPAINE Miljenko, Assistant
Geodetski Fakultet
Kaciceva, 26
Zagreb
Jugoslavia

LATERZA Luiz Bandeira de Mello, Assist. Prof.
Escola Politecnica da USP
Dept. de Construcao Civil
Caixa Postal 8174
Sao Paulo
Brasil

LJUBIC Vladislav, Researcher
University of Ljubljana
FAGG. Jamova, 2
P.O.Box 579
61000 Ljubljana
Jugoslavia

LOCURATOLO Elvira, Researcher
CNR
Istituto di Elaborazione dell'Informazione
Via Santa Maria, 46
56100 Pisa
Italy

MARCHETTI Alberto, Assoc. Professor
Universita' di Roma
Dip. di Informatica e Sistemistica
Via Buonarroti, 12
00185 Roma

NURMI Otto, Research Assistant
Institut für Angewandte Informatik
Universität Karlsruhe
Postfach 6380
D-7500 Karlsruhe
Western Germany

PAOLUZZI Alberto, Assoc. Professor
Universita' di Roma
Dip. di Informatica e Sistemistica
Via Buonarroti, 12
00185 Roma
Italy

PASCOLETTI Adriano, Assoc. Professor
Universita' di Udine
Ist. di Matematica, Informatica
e Sistemistica
Via Mantica, 3
33100 Udine
Italy

PELAGGI Antonia, Student
Universita' di Roma
Dip. di Informatica e Sistemistica
Via Buonarroti, 12
00185 Roma
Italy

RAUTU Sandu, Professor
Ben Gurion University
of the Negev
Dept. of Mechanical Engineering
Beersheva
Israel

SACK Jörg, Assistant Professor
Carleton University
School of Computer Sciences
Ottawa, K1S 5B6
Canada

SANTORO Esamuele, Researcher
Universita' di Salerno
Facolta' di Ingegneria
84100 Salerno
Italy

SINGH Hukum, Ph.D.
Technical Univ. of Budapest
Dept. of Geometry
Stoczek U. 4, H. II. 21
Budapest
Hungary

SLUZEK Andrzej, Doctor
Technical University of Warsaw
Institute of Automatic Control
ul. Nowowiejska, 15/19
00-665 Warszawa
Poland

STARITA Antonina, Assoc. Professor
Universita' di Pisa
Dip. di Informatica
Corso Italia, 40
56100 Pisa
Italy

STROTHOTTE Thomas, Postdoctoral Fellow
INRIA
Rocquencourt, B.P. 105
78153 Le Chesnay Cedex
France

TASSO Carlo, Assoc. Professor
Universita' di Udine
Ist. di Matematica, Informatica
e Sistemistica
Via Mantica, 3
33100 Udine
Italy

WIDMAYER Peter, Assistant Professor
Universität Karlsruhe
Inst. f. Angewandte Informatik
und Formale Beschreibungsverfahren
Postfach 6380
7500 Karlsruhe
Western Germania

WU Julian, Mathematician
U.S. Army
European Research Office
223-231 Old Marylebone Rd.
London NW1
United Kingdom

LECTURERS

HINRICHS Klaus,
ETH-Zentrum
Institut für Informatik
8092 Zürich
Switzerland

HOPCROFT John, Professor
Cornell University
Dept. of Computer Science
405 Upson Hall
Ithaca, N.Y. 14853
USA

LUCCIO Fabrizio, Professor
Universita' di Pisa
Dipartimento di Informatica
Corso Italia, 40
56100 Pisa
Italy

MEIER Andreas,
ETH-Zentrum
Institut für Informatik
8092 Zürich
Switzerland

OTTMANN Thomas, Professor
Universität Karlsruhe
Inst. f. Angewandte Informatik
Postfach 6380
7519 Karlsruhe
Western Germany

PREPARATA Franco, Professor
Coordinated Science Laboratory
University of Illinois
Urbana, Illinois 61801
USA

SERAFINI Paolo, Assoc. Professor
CISM and Universita' di Udine
Dip. di Matematica, Informatica
e Sistemistica
Via Mantica, 3
33100 Udine
Italy

STANAT Donald, Professor
University of North Carolina
Dept. of Computer Science
New West Hall 035a
Chapel Hill, NC. 27514
USA

—> Paolo [illegible]

16 14:15 1985 cism Page 1

tract for talks at CISM

Introduction to Algorithms and Data Structures

Donald F. Stanat
The University of North Carolina at Chapel Hill
Chapel Hill, North Carolina, USA

se talks will provide a fast-paced introduction to the field of
orithms and data structures. We will begin with a description of
basic abstract data structures: arrays, lists, trees and graphs,
ether with implementations of the structures. We will include
sures of cost, both in time and space, of various operations on
se structures for each of the implementations. Finally, we will
cuss some selected advanced data structures, including hash tables,
ps and a number of different kinds of balanced trees.

we will survey some general algorithm types, including greedy
orithms, divide and conquer, and exhaustive search, including some
the ways of making exhaustive search feasible: backtracking, branch
bound, and dynamic programming.

ally we will describe the cost of solving really hard problems and
roduce the notion of NP (Nondeterministic Polynomial) difficulty.
time permits, we'll also talk about approximation algorithms for
blems whose exact solutions have NP cost; these approximations
orithms provide a means of getting a non-optimal solution at
erate cost for a problem whose optimal solution is too expensive.

Storage and access structures for geometric data bases

J. Nievergelt and K. Hinrichs
Informatik, ETH, CH-8092 Zürich

Abstract

Geometric computation and data bases, two hitherto unrelated computing technologies, have begun to influence each other in response to the growing use of graphics and computer-aided design. CAD imposes a new challenge to data base implementers. A data base system for CAD must manage "in designer real time" large collections *not of points, but of spatial objects*, in such a way that proximity queries (such as intersection, contact, minimal tolerances) are answered efficiently. There are many techniques for reducing the problem of storing spatial objects to storing (sets of) points. Common to all of them is the problem that simple queries on objects turn into complex queries on points - much more complex than orthogonal range queries.

We describe in detail a technique which is particularly suitable for storing geometric objects built up from simple primitives, as they commonly occur in CAD. Proximity queries are handled efficiently as part of the accessing mechanism to disk. This technique is based on a *transformation of spatial objects into points in higher-dimensional parameter spaces*, and on the data structure *grid file* that answers region queries of complex shape efficiently. The grid file is designed to store higly dynamic sets of multi-dimensional data in such a way that common queries are answered using few disk accesses: a point query requires two disk accesses, a region query requires at most two disk accesses per data bucket retrieved. We describe a software package that implements the grid file and some of its applications.

Contents

1 Geometric computation and data bases

Geometric computation and data bases, two hitherto unrelated computing technologies, have begun to influence each other in response to the growing use of graphics and computer-aided design (CAD). We recall their origins, goals, typical techniques, and explain the difficulties each of them has in handling the requirements of the other.

1.1 Three generations of computing applications

The types of computer applications dominant at different times may be classified into three generations according to their influence on the development of computing.

. USING A RELATIONAL DBMS FOR SOLID MODELING

raditionally, engineering and design data has been handled by ad-hoc or simple file ystems with the inherent disadvantages of high redundancy and poor or non-existent data ndependence. In other words, the way the data is physically organized within the file must e known to the programs that access the data: Any change in the data organization requires hanging the programs and vice-versa. When the number of files increases, the consistent reatment of data becomes a problem in itself. It is not surprising therefore that today's levelopers of CAD systems begin to realize the importance of independent data organization [Ullman 1982]. They start experiments with databases although engineering applications exhibit characteristics which impose specific requirements on existing DBMS. n this section, we discuss how solids either by the CSG- or BR-approach may be stored in a elational database and list the main advantages and disadvantages.

.1. Constructive Solid Geometry

Although none of the existing solid representation schemes is suitable for all applications, the CSG-scheme provides a concise way to store a volumetric object. Halfspaces may be used as primitives at the lowest level. However, the resulting object representations are not necessarily regular sets [Requicha 1980] because of the unboundedness of primitive halfspaces. Instead, cubes, cones, cylinders etc. are usually used as primitives. A 3D object can then be described with the following grammar:

```
<OBJECT> ::=        <PRIMITIVE> |
                    <OBJECT> <MOTION> ARGUMENT |
                    <OBJECT> <OPERATION> <OBJECT>
<PRIMITIVE> ::=     CUBE | CYLINDER | CONE | SPHERE | ...
<MOTION> ::=        TRANSLATE | ROTATE | SCALE
<OPERATION> ::=     UNION | INTERSECTION | DIFFERENCE
```

The semantics of a CSG-representation is clear: Each subtree represents a solid, i.e. a regular set, resulting from the combinatorial or motion operators to the subparts. The dynamic behavior of the data structure results from the recursiveness embedded in the grammar rules. In the relational model, recursiveness has to be broken down, and a solid may be described by several relations: It has to be treated as a whole at a high object level while providing its individual details at lower part levels.

The conceptual scheme of the CSG-approach is given in Fig. 2: Each OBJECT in a CSG-representation consists of several PARTS. In the relational model, this hierarchical structure has to be described by at least two independent relations in order to limit data redundancy. Furthermore, two generic structures are imposed by the CSG-scheme [Lee and Fu 1983] which cannot be defined directly by relations. First, each part may either be a TRANSFORMED-, or a PRIMITIVE-, or a COMBINED-PART according to the grammar rules, for instance, a COMBINED-PART consists of two parts, namely the FIRST one and the SECOND one plus the corresponding Boolean OPERATION union, intersection, or difference. Second, each primitive part is either a CUBE, or a CONE, or a CYLINDER etc.

orientability, i.e. faces may intersect only at common edges or vertices, and each edge is shared by exactly two faces etc. Mathematically, the surface of a solid described in boundary representation may be treated as a manifold.

After having discussed the main schemes for representing solids we ask which ones are suitable for database techniques:

Fig. 1: Using Database Techniques for Solid Modeling.

In *Primitive Instancing* it is obvious that each instance may be considered as a record or tuple in a Data Base Management System (DBMS). Since a shape type and a limited set of parameter values specify an object, parametrization does not involve much work for geometric and topological computation. Therefore, every commercial DBMS might be good enough for describing and storing a part family. On the other hand, both *Spatial Enumeration* and *Cell Decomposition* schemes are not adequate using database techniques, especially if the objects are described in fine resolution or by a large number of cells. The cost of database interaction for object manipulation becomes unreasonably high. Storing objects described by *Constructive Solid Geometry* or *Boundary Representation* schemes in a database, however, seems promising.

This paper will concentrate on database aspects for solid modeling. Section 2 describes how objects given in CSG- or BR-representation may be mapped into a relational database scheme. In section 3, a surrogate model is introduced to better support geometric and topological information. A proposal for direct handling of vectors, matrices, and tensors in the relational model is outlined in section 4. First experiments and conclusions are given in section 5.

1. REPRESENTATION SCHEMES FOR SOLIDS

For Computer-Aided Design (CAD) of three-dimensional objects (3D objects or solids), the geometric and topological aspects of part and assembly specification is important. In [Requicha 1980], several representation schemes for solids are discussed some of which we briefly describe.

Primitive Instancing is based on a family or group of objects where each member is distinguishable by a few parameters. For instance, the family of cog wheels may be described by a type code, the wheel's diameter and the number of equally spaced cogs. Other properties of the objects are not specified explicitly; they either are constant throughout the family or they depend on specified parameters. Primitive instancing lacks the possibility of combining representations in order to create new or more complex schemes. It is also difficult or even impossible to derive geometric and topological properties directly from such schemes. In practice however, parametrization is applicable (and still widely used) as long as the catalog of parameters does not become to large.

Spatial Enumeration denotes a scheme where the embedding space is divided into a grid of volume elements, and a solid is represented by a list of occupied grid blocks or elements. Recently, the octree encoding (e.g. [Meagher 1982]) as a hierarchical spatial enumeration has been discussed as a representation scheme for solid modeling. It divides the space occupied by a solid into eight cubic parts recursively until a fixed maximal resolution is reached. There are some advantages to this data structure; e.g. Boolean operations, hidden surface removal or interference detection show linear growth because all objects are kept spatially pre-sorted at all times. However, when moving objects are taken into account, more computation is involved.

Cell Decomposition methods are based on the results of triangulation theory. A solid or polyhedron is decomposed into disjoint parts of different dimensions. Therefore, operations and calculations become easier due to disjointness. Cell decomposition may be considered as a generalized spatial occupancy enumeration where cells neither have to lie on a fixed grid nor have a pre-specified size and shape. In [Bieri and Nef 1982], a recursive sweep-plane algorithm is presented that enumerates the cells of all dimensions into which space may be partitioned by a finite set of hyperplanes. The described method is also suitable to compute the Euler characteristic, the volume or other integral parts of polyhedrons represented in Boolean form. Local information is collected at every vertex and summed up for the result.

Constructive Solid Geometry (CSG) denotes a family of representing schemes where each object is described as Boolean construction or combination of solid components via the regularized set operations such as union, intersection, and difference. Regularity provides a natural formalization of dimension preserving properties [Tilove 1980], i.e. the result of a Boolean operation of two solids is volumetric; dangling edges and faces or isolated points are not allowed. It is important to note that each CSG-scheme may be described as a tree where non-terminal nodes represent operations both for construction and transformation, and terminal nodes denote primitives or arguments of motion respectively.

Boundary Representation (BR) describes a solid by its bounding surface which often is subdivided into curvature-continous regions known as faces. Each face as a region of its underlying surface is again bounded by a perimeter ring of edges which are, in turn, bounded by a pair of vertices. The bounding surface has some unique characteristics such as

APPLYING RELATIONAL DATABASE TECHNIQUES TO SOLID MODELING+)

Andreas Meier
Informatik, ETH Zurich
CH-8092 Zurich

Abstract:
Two main approaches to solid modeling have been taken by developers of CAD systems. One is to rely on a set of primitives and to use regularized set operations (i.e. Constructive Solid Geometry), the other is to rely on a set of Euler operators that combine faces, edges, and vertices (i.e. Boundary Representation). Investigating both approaches, we discuss some of the shortcomings when storing geometric objects in a relational database. In addition, we describe a surrogate concept currently being implemented which allows the user to define structural relationships among semantically related data. Based on surrogate values, two constructs PART-OF and IS-A are defined in order to retrieve and manipulate geometric objects efficiently. Finally, a structured type for handling vectors, matrices, and tensors as attribute values is proposed by dropping the First Normal Form.

Keywords:
Geometric modeling, constructive solid geometry, boundary representation, relational database, surrogates, vectors, matrices, tensors.

Contents:

+) This is a revised version of the paper given at the GI-Conference on Database Systems for Office Automation, Engineering, and Scientific Applications, Karlsruhe, W. Germany, March 1985. This work was supported, in part, by the Swiss National Science Foundation, under grant number 2.533-0.82.

been registered. Besides the coordinates of the corresponding hectare there are other attributes stored in each record, for instance the identification number of the municipality the hectare belongs to or the type of ground cover of the hectare; since these attributes are not used as keys for performing queries they are neglected. The records have been inserted into a two-dimensional grid file using as keys the two coordinates of the corresponding hectare. Typically, these records are accessed by range queries to find all the hectares that belong to a rectangular region.

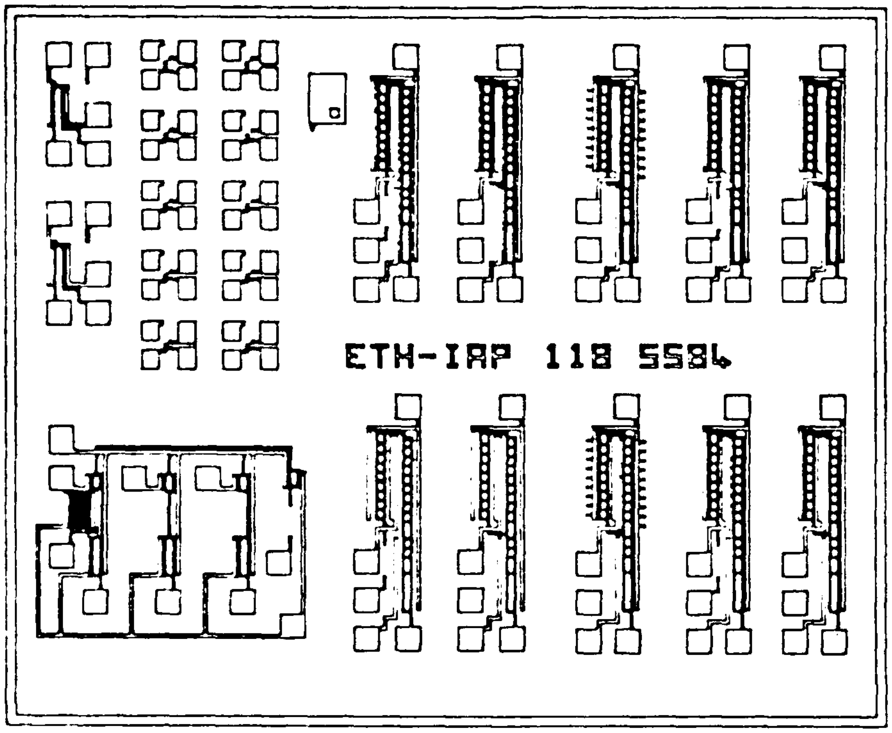


Fig. 2.7: Connected components of a layout mask.

References

[And 83] D. P. Anderson: *Techniques for reducing pen plotting time*, ACM Trans. Graphics 2, 3 (1983), 197-212.

[Hin 85a] K. Hinrichs: *The grid file system: implementation and case studies of applications*, Diss. ETH No. 7734, 1985.

[Hin 85b] K. Hinrichs: *Implementation of the grid file: design concepts and case studies*, to appear in BIT.

[MKY 81] T. H. Merrett, Y. Kambayashi, H. Yasuura: *Scheduling of page-fetches in join operations*, Proc. 7th Intern. Conf. on Very Large Data Bases, Cannes, France (1981), 488 - 497.

[NHS 84] J. Nievergelt, H. Hinterberger, K. C. Sevcik: *The grid file: an adaptable, symmetric multikey file structure*, ACM Trans. on Database Systems 9, 1 (1984), 38 - 71.

Portability has been achieved by defining two interfaces: One towards the host (hardware and operating system), the other towards client programs. The software can be transferred to other systems by adapting a module GFHost that isolates the machine- and disk-dependent parts of the program. It provides procedures for:

- creating and initializing disk storage;
- opening and closing communication channels between the disk and the grid file module;
- creating, deleting, reading and writing disk blocks;
- managing empty disk blocks.

The interface towards client programs consists of several modules that provide utility and query procedures:

- creating, deleting, opening and closing a grid file.
- inserting and deleting records in a grid file.
- changing non-key information in a record.
- point query: find all records with given key values $x_1, \ldots, x_k$ (if keys are unique at most one record will be found).
- range query: find all records whose key values x_i lie in given intervals $[l_i, u_i]$ $(1 \le i \le k)$.
- user defined region query: the user has to write a procedure which is called by the *grid file system* and determines whether a grid cell (given by intervals $[l_i, u_i)$ $(1 \le i \le k)$) intersects the search region defined by the user.
- nextabove, nextbelow: given key i with key value x_i, find the records with key values above or below x_i and next to x_i; this gives the user the possibility to process the records sequentially with respect to one key.
- join query: the join query is a generalization of the join operator known from relational data bases. The user has to write some procedures which are called by the *grid file system* and guide the join query.
- counting: the above queries can be performed by only counting the records, but not transferring them to the user program.

2.5 Case studies of applications

The grid file software package has been used to store and process geometric objects in the following applications [Hin 85a].

Producing layout masks for integrated circuits (Fig. 2.7). Mask generated by a CAD system for chip design (David Mann Format) are presented as a set of aligned rectangles. Fabrication requires that a mask is represented as the *set of connected components* generated by rectangle overlap, i. e. a set of aligned polygons (all edges parallel to the coordinate axes, Manhattan geometry). This transformation program was implemented by processing rectangles in a 4-dimensional grid file and computing the connected components by intersection queries.

Preprocessing plotter files. In a CAD system for mechanical engineering plotter files are preprocessed in order to reduce the total distance along which the raised pen has to be moved. The task of finding an optimal solution to this problem is equivalent to the traveling salesman problem and therefore NP-complete. The plotter files contain line segments and arcs which have to be drawn. The end points of the line segments and the arcs are stored in a 2-dimensional grid file. A reduction of the total pen plotting time is achieved by nearest neighbor queries on this grid file. A similar method using quad trees is presented in [And 83].

Analyzing photographic satellite data. A photograph obtained by a satellite consists of 512 • 512 pixels. Each pixel is assigned four color values in the range from 0 to 255. These pixels are stored in a 4-dimensional grid file. The ground imaged by these pixels is then classified into water, forest, fields, residential and metropolitan areas etc., by range queries on this grid file.

Managing simple spatial objects. An interactive program manages large sets of simple spatial objects, e. g. rectangles, circles and segments. These objects, each of which is defined by a fixed number of parameters, are stored in different grid files, one for each type of object. The program allows the user to insert and delete simple spatial objects and to perform proximity queries (e. g. intersection, containment) on the stored data.

Processing geographic data. The Swiss Federal Office for Statistics made available to us a file which contains raster information about Switzerland. Each record in this file represents a square of 100 meters by 100 meters (1 hectare). Switzerland covers about 4'000'000 hectares, but only about 100'000 hectares have

2.3 Evaluating region queries with a grid file

We have seen that proximity queries on spatial objects lead to search regions significantly more complex than orthogonal range queries. The grid file [NHS 84] is a structure for storing multidimensional point data designed to allow the evaluation of irregularly shaped search regions in such a way that *the complexity of the region affects CPU time but not disk accesses*. The latter limit the performance of a data base implementation.

The grid file *partitions space* into raster cells and *assigns data buckets to cells*. The partition information is kept in *scales*, one for each axis of space; the assignment is recorded in an array called *grid directory*. The directory is likely to be large and must therefore be kept on disk, but the scales are small and can be kept in central memory. Therefore, the grid file realizes the *two-disk-access principle* for single point retrieval (exact match query): by searching the scales, the k coordinates of a data point are converted into interval indices without any disk accesses; these indices provide direct access to the correct element of the grid directory on disk, where the bucket address is located. In a second access the correct data bucket (i.e. the bucket that contains the data point to be searched for, if it exists) is read from disk. A query region Q is matched against the scales and converted into a set I of index tuples that refer to entries in the directory. Only after this preprocessing do we access disk to retrieve the correct pages of the directory and the correct data buckets whose regions intersects Q (Fig. 2.6).

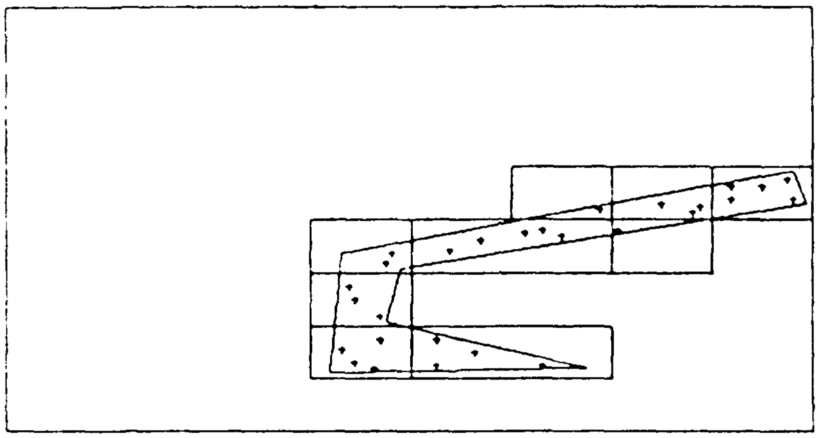


Fig. 2.6: Query in a grid file.

A geometric join query is answered in an analogous way. Let f and f' be the two grid files involved, and let H and H' be the underlying higher-dimensional spaces. The scales of f and f' define a grid on the Cartesian product $H \times H'$. The cells of this grid which intersect the search region in $H \times H'$ are determined by matching the scales of the two grid files against the search region. As in the case of proximity queries on a single grid file this computation needs no access to disk. If a cell intersects the search region the corresponding pair of buckets $(B_f, B_{f'})$ is accessed from disk via the grid directories of f and f'. If the Cartesian product of the bucket regions of B_f and $B_{f'}$ is completely contained in the search region all pairs of objects corresponding to pairs of points $(p_f, p_{f'})$ with $p_f \in B_f$ and $p_{f'} \in B_{f'}$ fulfill the join condition. If the Cartesian product of the bucket regions of B_f and $B_{f'}$ is not completely contained in the search region all pairs of points $(p_f, p_{f'})$ with $p_f \in B_f$ and $p_{f'} \in B_{f'}$ must be checked in order to see whether they lie inside or outside the search region, i. e. whether the corresponding pairs of objects fulfill the join condition. A buffer of minimal size of two pages receives pairs of data buckets $(B_f, B_{f'})$ according to a scheduling policy similar to the one mentioned in [MKY 81].

2.4 The grid file software package

The grid file is implemented in Modula-2 and FORTRAN-77 as a portable data management package of about about 5800 lines of source code. The Modula-2 version [Hin 85a, 85b] runs on the DEC-VAX 11 under VMS, on the DEC-PDP 11 under RT-11 and on some personal computers based on the Motorola 68000 processor. The FORTRAN-77 version has been developed on a DEC-VAX 11. The package includes a Prolog interpreter that gives the user interactive access to the data store ' in grid files, and serves as a powerful query language that permits deduction.

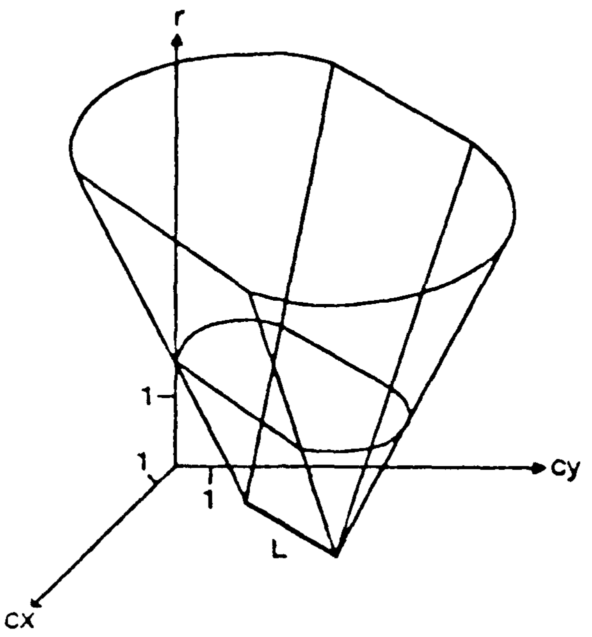


Fig. 2.4b: Search region for an intersection query with a line L.

Geometric join query. Let Ω' be another class of simple spatial objects with parameter space H', and $\Gamma \subset \Omega'$. For every $A \in \Omega$ let $H'_A \subset H'$ be the set of all points in H' representing $A' \in \Omega'$ such that A and A' intersect. Denote by P_A the point in H representing a spatial object $A \in \Omega$. The region in the Cartesian product $H \times H'$ that contains all points representing pairs $(A, A') \in \Gamma \times \Gamma'$ of intersecting objects is the union of the sets $\{P_A\} \times H'_A$ for all $A \in \Omega$; this region is particularly simple for the different classes of simple spatial objects.

Let Ω be the class of points, Ω' the class of intervals on a straight line. Then $H \times H'$ is the 3-dimensional space. All pairs (p, I) of points p with coordinate x and intervals I = (cx, dx) such that $p \in I$ are represented by points lying in the solid shown in Fig. 2.5. This solid is obtained by moving the search region for a point-in-interval query along the bisector in the x-cx-plane.

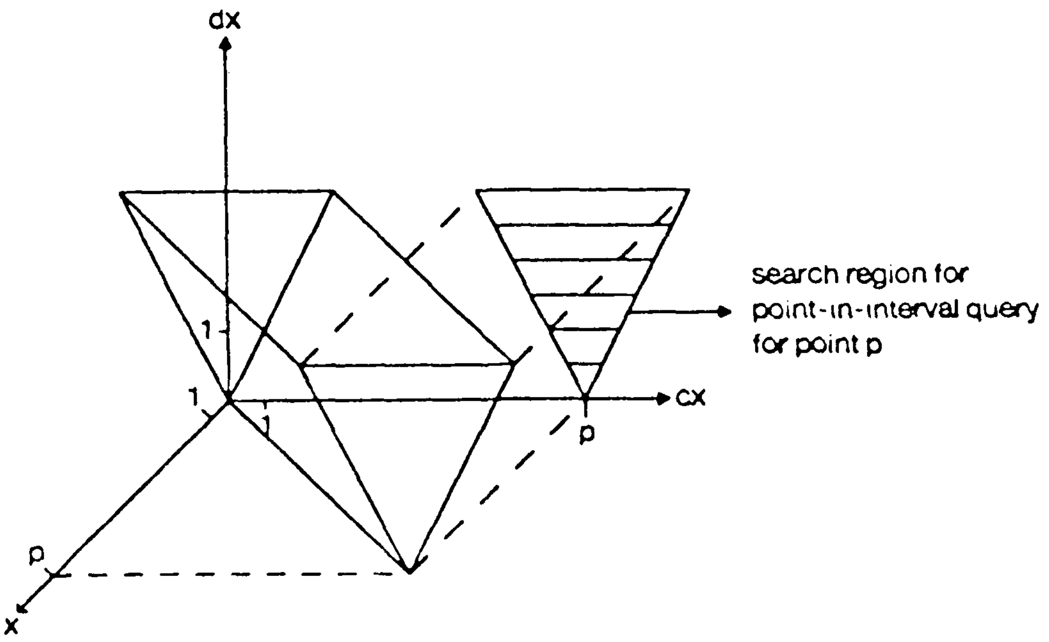


Fig. 2.5: Search region for a geometric join query.

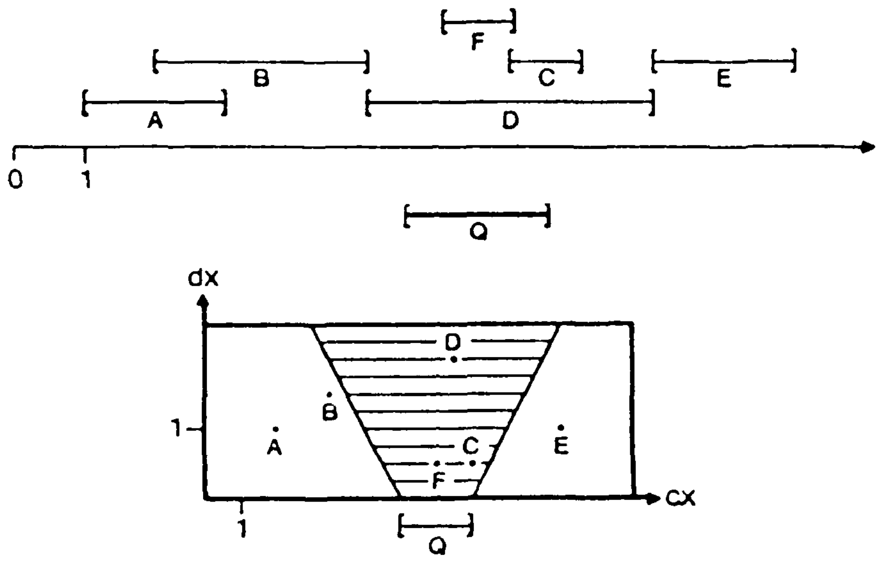


Fig. 2.3: Search region for an interval intersection query.

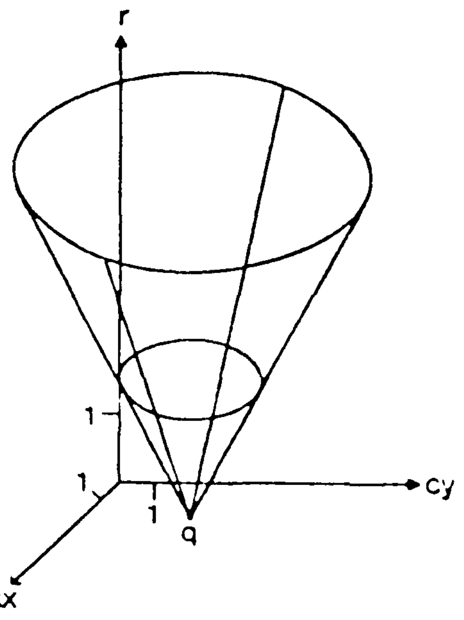


Fig. 2.4a: Search region for a point query in the class of circles in the plane.

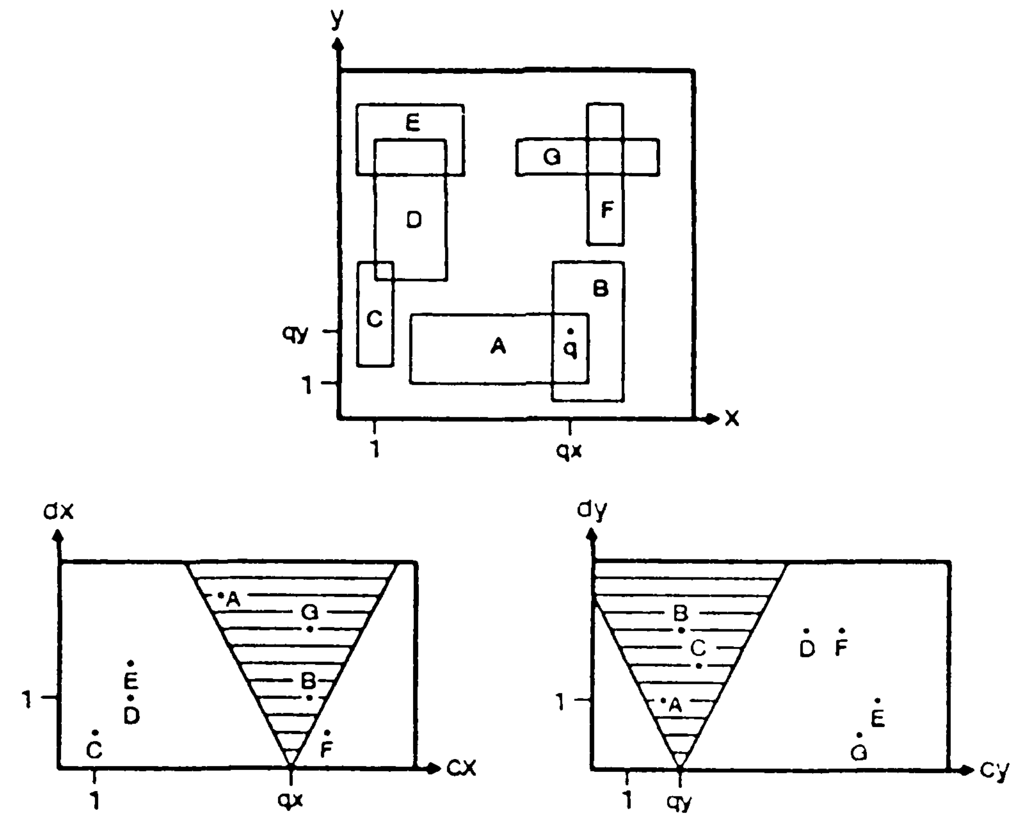


Fig. 2.2: Search region for a point query in the class of aligned rectangles in the plane.

3) Let Ω be the class of circles in the plane. As parameters for the representation of a circle as a point in 3-dimensional space we choose the coordinates of its center (cx, cy) and its radius r. All circles which overlap a point q are represented in the corresponding 3-dimensional space by points lying in the cone with vertex q shown in Fig. 2.4a. The axis of the cone is parallel to the r-axis (the extension parameter), its vertex is q considered as a point in the cx-cy-plane (the subspace of the location parameters).

Point set query. Given a set Q of points, the region in H that contains all points representing objects A $\in \Gamma$ which intersect Q is the *union* of the regions in H that result from the point queries for each point in Q. The *union of cones* is a particularly simple region in H if the query set Q is a simple spatial object.

1) Let Ω be the class of intervals on a straight line. An interval I = (cx, dx) intersects a query interval Q = (cq, dq) if and only if its representing point lies in the shaded region shown in Fig. 2.3; this region is given by the inequalities $cx - dx \leq cq + dq$ and $cx + dx \geq cq - dq$.

2) Let Ω be the class of aligned rectangles in the plane. If Q is also an aligned rectangle then Ω is again treated as the Cartesian product of two classes of intervals, one along the x-axis, the other along the y-axis. All rectangles which intersect Q are represented by points in 4-dimensional space lying in the Cartesian product of two interval intersection query regions.

3) Let Ω be the class of circles in the plane. All circles which intersect a line segment L are represented by points lying in the cone-shaped solid shown in Fig. 2.4b. This solid is obtained by embedding L in the cx-cy-plane, the subspace of the location parameters, and moving the cone with vertex at q along L.

intervals on a long line clustering along the diagonal, leaving large regions of a large embedding space unpopulated; whereas the same set of intervals represented by a location parameter cx and an extension parameter dx, fills a smaller embedding space in a much more uniform way. With the assumption of bounded d, this data distribution is easier to handle.

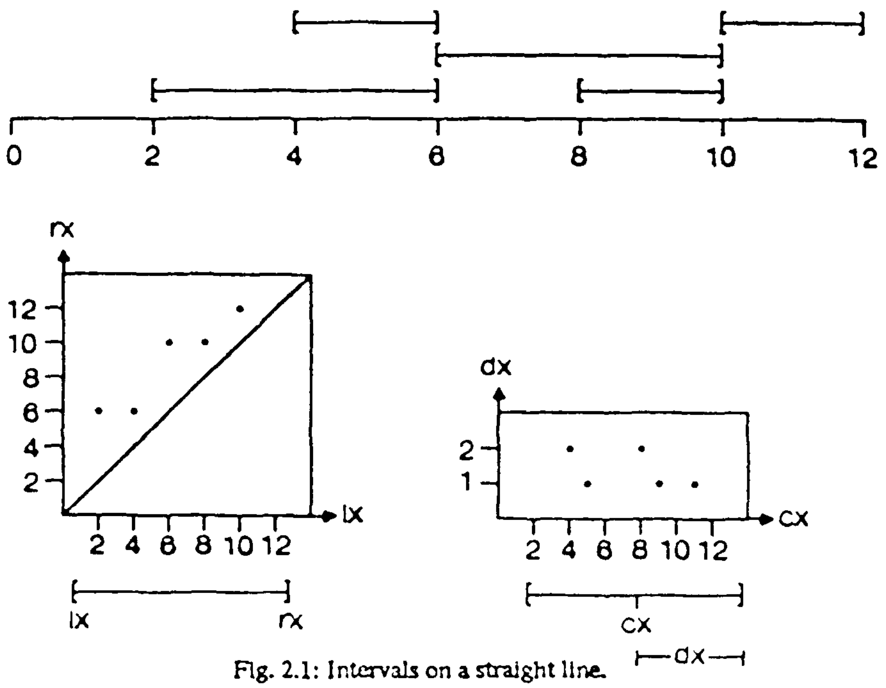


Fig. 2.1: Intervals on a straight line.

2.2 Intersection queries lead to cone-shaped search regions

Intersection is a basic component of other proximity queries, and thus deserves special attention. CAD design rules, for example, often require different objects to be separated by some minimal distance. This is equivalent to requiring that objects surrounded by a rim do not intersect. Given a class Ω of simple spatial objects with parameter space H, and a set $\Gamma \subset \Omega$ of simple objects represented as points in H, we consider three types of queries:

- point query:	given a query point q, find all objects $A \in \Gamma$ for which $q \in A$.
- point set query:	given a set Q of points, find all objects $A \in \Gamma$ which intersect Q.
- geometric join query:	given another class Ω' of spatial objects with parameter space H', and a set $\Gamma' \subset \Omega'$, find all pairs $(A, A') \in \Gamma \times \Gamma'$ of intersecting objects.

Point query. For a query point q compute the region in H that contains all points representing objects in Γ which overlap q.

1) Let Ω be the class of intervals on a straight line. An interval given by its center cx and its half length dx overlaps a point q with coordinate qx if and only if $cx - dx \leq qx \leq cx + dx$.

2) The class Ω of aligned rectangles in the plane (with parameters cx, cy, dx, dy) can be treated as the Cartesian product of two classes of intervals, one along the x-axis, the other along the y-axis. All rectangles which contain a given point q are represented by points in 4-dimensional space lying in the Cartesian product of two point-in-interval query regions (Fig. 2.2). The region is shown by its projections into the cx-dx-plane and the cy-dy-plane.

simple object such as a point or a line segment. Even if the solid and the query are far apart, *all the components of the solid must be examined in a tree traversal to detect this.* What is lacking is some concisely stated geometric information that describes global properties of the solid and its location in space.

2 An approach to combined geometric modeling and storing: Approximation, transformation to parameter space, grid file

The technique we now present for modeling and storing spatial objects is based on 1) *approximation of complex spatial objects by simple shapes, e.g. containers,* 2) *transformation of simple spatial objects into points in higher-dimensional parameter spaces,* and 3) *the grid file for point storage.*

Complex, irregularly shaped spatial objects can be represented or approximated by simpler ones in a variety of ways, for example: *decomposition,* as in a quad tree tessellation of a figure into disjoint raster squares of size as large as possible; representation as a *cover* of overlapping simple shapes; *enclosing it in a container* chosen from a class of simple shapes. The container technique allows efficient processing of proximity queries because it preserves the most important properties for proximity-based access to spatial objects, in particular: it does not break up the object into components that must be processed separately, and it eliminates many potential tests quickly (if two containers don't intersect, the objects within won't either). As an example, consider finding all polygons that intersect a given query polygon, given that each of them is enclosed in a simple container such as a circle or an aligned rectangle. Testing two polygons for intersection is an expensive operation as compared to testing their containers for intersection. The cheap container test excludes most of the polygons from an expensive, detailed intersection check.

Any approximation technique limits the primitive shapes that must be stored to *one or a few types,* for example aligned rectangles or boxes. An instance of such a type is determined by a few parameters, such as coordinates of its center and its extension, and can be considered to be a point in a (higher-dimensional) parameter space. This transformation reduces object storage to point storage, increasing the dimensionality of the problem but *without loss of information.* Combined with an efficient multidimensional data structure for *point* storage it is the basis for an effective implementation of data bases of spatial objects.

2.1 Transformation to parameter space

Consider a class of simple spatial objects, such as aligned rectangles in the plane (i.e. with sides parallel to the axes). Within its class, each object is defined by a small number of parameters. For example, an aligned rectangle is determined by its center (cx, cy) and the half-length of each side, dx and dy.

An object defined within its class Ω by k parameters, can be considered to be a point in a k-dimensional parameter space H assigned to Ω. For example, an aligned rectangle becomes a point in 4-dimensional space. *All of the geometric and topological properties of an object can be deduced from the class it belongs to and from the coordinates of its corresponding point in parameter space.*

Different choices of the parameter space H for the same class Ω of objects are appropriate, depending on characteristics of the data to be processed. Some considerations that may determine the choice of parameters are:

1) Distinction between *location parameters* and *extension parameters.* For some classes of simple objects it is reasonable to distinguish location parameters, such as the center (cx, cy) of an aligned rectangle, from extension parameters, such as the half sides dx and dy. This distinction is always possible for objects that can be described as Cartesian products of spheres of various dimensions. For example, a rectangle is the product of two 1-dimensional spheres, a cylinder the product of a 1-dimensional and a 2-dimensional sphere. Whenever this distinction can be made, cone-shaped search regions generated by proximity queries as described in section 2.3 have a simple intuitive interpretation: The subspace of the location parameters acts as a "mirror" that reflects a query.

2) *Independence of parameters, uniform distribution.* As an example, consider the class of all intervals on a straight line (Fig. 2.1). If intervals are represented by their left and right endpoints, lx and rx, the constraint lx $\leq$ rx restricts all representations of these intervals by points (lx, rx) to the triangle above the diagonal. Any data structure that organizes the embedding space of the data points, as opposed to the particular set of points that must be stored, will pay some overhead for representing the unpopulated half of the embedding space. A coordinate transformation that distributes data all over the embedding space leads to more efficient storage. The phenomenon of nonuniform data distribution can be worse than this. In most applications, the building blocks from which complex objects are built are much smaller than the space in which they are embedded, as the size of a brick is small compared to the size of a house. If so, parameters such as lx, rx that locate boundaries of an object, are highly dependent on each other. Fig. 2.1 shows short

1.3 The conventional data base approach to "non-standard" data

Data base technology has developed over the past two decades in response to the needs of commercial data processing. The key concepts introduced and supported by data base software mirror the reality that used to be handled manually by office clerks. Large quantities of records of a few different types, identified by a small number of attributes, mostly retrieved in response to relatively simple queries: point queries that ask for the presence or absence of one particular record, interval or range queries that ask for all records whose attribute values lie within given lower and upper bounds. More complex queries tend to be reduced to these basic types.

Data base software has yet to take into account the specific requirements of geometric computation, as can be seen from the terminology used: Geometric objects are lumped into the amorphous pool of "non-standard" applications. The sharp distinction between the *logical view* presented to the user and the *physical aspects* that the implementer sees has been possible in conventional data base applications because data structures that allow efficient handling of point sets are well understood. The same distiction is premature for geometric data bases: in interactive applications such as CAD efficiency is the real issue, and until we understand geometric storage techniques better we may not be able to afford the luxury of studying geometric modeling divorced from physical storage. Consider the following example.

A set of polyhedra might be stored in a relational data base by using the boundary representation (BR) approach: a polyhedron p is given by its faces, a face f by its bounding edges, an edge e by its endpoints s_1 and s_2. Four relations *polyhedra*, *faces*, *edges* and *points* might have the following structure:

A tuple in the relation *polyhedra* is a pair (p_i, f_k) of identifiers for a polyhedron and a face: f_k is a face of polyhedron p_i.

A tuple in the relation *faces* is a pair (f_k, e_j) of identifiers for a face and an edge: e_j is a bounding edge of face f_k.

A tuple in the relation *edges* is a triple (e_j, s_l, s_m) of identifiers for one edge and two points: s_l and s_m are the endpoints of edge e_j.

A tuple in the relation *points* is a triple (s_n, x, y): s_n is the identifier of a point, x and y are its coordinates.

This representation smashes an object into parts which are spread over different relations and therefore over the storage medium. The question whether a polyhedron P intersects a given line l is answered by intersecting each face f_k of a polyhedron p_i with l. If the tuple (p_i, f_k) in the relation *polyhedra* contains the equation of the corresponding plane, the intersection point of the plane and the line l can be computed without accessing other relations. But in order to determine whether this intersection point lies inside or outside the face f_k requires accessing tuples of *edges* and *points*, i. e. accessing different blocks of storage, resulting in many more disk accesses than the geometric problem requires.

1.4 Geometric modeling separated from storage considerations

In this early stage of development of geometric data base technology, we cannot afford to focus on modeling to the exclusion of implementation aspects. *In graphics and CAD the real issue is efficiency*: 1/10-th of a second is limit of human time resolution, and a designer works at maximal efficiency when "trivial" requests are displayed "instantaneously". This allows a couple of disk accesses only, which means that geometric and other spatial attributes must be part of the retrieval mechanism if common geometric queries (intersection, inclusion, point queries) are to be handled efficiently.

A key problem that affects efficiency is how to reduce complex objects to simpler ones chosen from predefined primitives. Among the standard techniques known we have already discussed how boundary representations stored in a relational data base prevent efficient access based on geometric queries. The problem of an object being torn apart happens also in another standard modeling technique, *constructive solid geometry* CSG. Let us briefly discuss the consequences of basing the physical storage structure directly on such modeling techniques.

In constructive solid modeling a complex object is constructed from simple primitives, such as cubes or spheres, by means of Boolean operations union, intersection and difference. The construction process is represented by a tree. Each leaf of a CSG tree contains a simple object, each internal node contains a Boolean operation. To each node a geometric transformation such as scaling, translation and rotation may be assigned. The Boolean operation is applied to the objects represented by the left and right subtree of the node. A geometric transformation assigned to a leaf is applied to the simple object stored in the leaf, a geometric transformation assigned to an internal node is applied to the object resulting from the Boolean operation stored in this node. Now consider the query whether a solid in CSG representation intersects a

The first generation, characterized by *numerical computing*, led to the development of many new *algorithms*. It transformed numerical analysis from a craft to be practiced by every applied mathematician into a field for specialists. It soon became obvious that writing good (efficient, robust) numerical software requires so much knowledge and effort that this task cannot be left to the applications programmer. The development of large portable numerical libraries became one of the major tasks for professional numerical analysts.

The second generation, hatched by the needs of commercial data processing, led to the development of many new *data structures*. It focused attention on the problem of *efficient management of large, dynamic data collections*, initially under batch processing conditions. *Searching and sorting* were recognized as basic operations whose time requirements turned out to be the bottleneck for many applications. *Data base technology* emerged to shield the end-user from the details of implementation (storage techniques, features dependent on hardware- and operating system), by presenting the data in the form of *logical models* that highlight *relationships among data items* rather than their internal representation, and by introducing the abstraction of *access path* to hide detailed access algorithms of underlying data structures.

We are now on the threshold of a third generation of applications, dominated by *computing with pictorial and geometric objects*. This change of emphasis is triggered by today's ubiquitous interactive use of personal computers, and their increasing graphics capabilities. It is a simple fact that people absorb information fastest when it is presented in pictorial form, hence *computer graphics and the underlying processing of geometric objects will play a role in the majority of computer applications*. The field of *computational geometry* has emerged as a scientific discipline during this past decade in response to the growing importance of processing pictorial and geometric objects. It has already created novel and interesting algorithms *and* data structures, and is beginning to impact data base technology under the label (hopefully temporary) of *non-standard database applications*. In order to understand *how* geometric computation is likely to affect data bases, it is useful to survey some milestones in this rapidly developing field which is replacing the traditional areas of numeric computation and of data management as the major research topic in algorithm analysis.

1.2 Computational geometry - theory and practice

During the seventies geometric problems caught the attention of researchers in concrete complexity theory. They brought to bear the finely honed tools of algorithm analysis and achieved rapid progress. Elementary problems (e. g. determining intersections of simple objects such as line segments, aligned rectangles, polygons) yielded elegantly to general algorithmic principles such as divide-and-conquer or plane sweep. But in many instances a surprisingly large increase of difficulty showed up in going from two to three dimensions: for example, intersection of polyhedra is still an active research topic where major efficiency gains are to be expected. The *theory of computational geometry*, although well underway, has as yet explored only a fraction of its potential territory.

The *practice of computational geometry* is even less well understood. Many important geometric problems in computer-aided design, in geographical data processing, in graphics do *not* lend themselves to being studied and evaluated by the asymptotic performance formulas that the algorithm analyst cherishes. For example, asymptotics does not help in answering the question whether we can access an object in one disk access or two, thus being able to display it "instantaneously" on the designer's screen - realistic assumptions about the size of today's central memories are needed. Nor will asymptotics settle the argument raging in the CAD community between proponents of boundary representations and adherents of constructive solid geometry - taste, experience, and type of application are the relevant parameters. And below the highly visible issues of object representation, data structures and algorithms hide the tantalizing details of the *numerics* of computational geometry, such as the problems caused by "braiding straight lines", which may intersect repeatedly.

Commercially available software in computer graphics and CAD has not yet taken into account the results of computational geometry. Straightforward algorithms are being used whose theoretical efficiency is poor as compared to known results. *Perhaps the straightforward algorithms are better in practice than theoretically optimal ones, but such difficult questions have hardly been investigated*, as CAD systems development today is so labor intensive that all resources are absorbed by just getting the system to work, and algorithm analysis has so far largely restricted itself to theoretically measurable performance.

We know by analogy with numerical analysis what the next step should be in the maturing process of computational geometry: *The development of efficient, portable, robust program libraries for the most basic, frequent geometric operations on standard representations of geometric objects*. In other words, we must develop the geometric subroutine library of CAD, thus exposing theoretical results to stringent practical tests.

Fig. 2: Conceptual Scheme of the CSG-Approach.

In the classical relational model, data is organized as record instances (tuples) in tables (relations). There are no schema defined relationships such as PART-OF and IS-A structures. Instead, a high level language (predicate calculus or relational algebra) is used for exploiting relationships based on values. However, the mapping of highly interrelated data into tuples in one or more relations has to be done entirely by the user. For instance, the important feature of a *generic structure* which says that all descendant objects must bear the same key domains as their ascendants has to be enforced by the user himself. The relational model does not allow individual objects to be uniformly referred to regardless of the generic class in which they appear.

Other drawbacks using the classical relational model are due to normalization. A relation is said to be in *First Normal Form* [Ullman 1982] if and only if it satisfies the constraint that it contains atomic values only. Note that in our example of a CSG-scheme, this condition is very inconvenient. For instance, a TRANSFORMED-PART may be described by a part number and a transformation matrix, e.g. a 4x4-matrix in homogeneous coordinates. To describe these facts in a normalized relation, sixteen attributes must be introduced artificially which correspond to the matrix arguments. Of course, one would instead only define three arguments PARAM1, PARAM2 and PARAM3 for motion parameters, plus a further attribute MOTION which denotes if it is a translation, a rotation, or a scaling operation. In any case, the First Normal Form is cumbersome.

In conclusion, the study of the CSG-approach in solid modeling suggests the following extensions to the relational model: There should be a way of defining PART-OF and IS-A structures explicitly to the system in order to give the user the possibility of querying an object or part of it as a whole rather than assembling different relations and thinking about all known interrelationships. In addition, the First Normal Form should be dropped. Or at least, the user should have a direct way of storing matrices as data types into a tuple, and the database system should incorporate features for non-atomic fields into its calculus or algebra.

2.2. Boundary Representation

With the Boundary Representation model, solids are described by a collection of faces which in turn are represented by their bounding edges and faces. So called *Euler operators* allow incremental manipulation of the objects while restoring well-formedness of the surfaces: closedness, orientability, nonself-intersection, boundedness, and connectivity [Eastman and Weiler 1979]. These topological properties are condensed in the Euler-Poincaré formula: with f faces, e edges, v vertices, r inner loops of faces called rings, c cavities or hollow tubes, and g holes through the body (or genus g corresponding to the number of handles in graph topology) the following condition holds:

$$f - e + v - r = 2 \bullet (c - g)$$

The practical relevance of the formula is ensuring that shapes are topologically well-formed; e.g. its application eliminates the danger of ill-formed solids such as the Klein bottle. If we consider the above formula as a hyperplane in six-dimensional space, the law restricts the valid transitions to a subset of all those combinatorically possible. Of course, the desired set of Euler operators should cover the hyperplane; a possible spanning set of five primitive operators may be defined as follows:

	f	e	v	r	c	g
MEF resp. KEF	1	1	0	0	0	0
MEV resp. KEV	0	1	1	0	0	0
MEKR resp. KEMR	0	1	0	-1	0	0
MFVC resp. KFVC	1	0	1	0	1	0
MFKGR resp. KFMGR	1	0	0	-1	0	-1

The Euler operator MEF stands for "Make Edge and Face" which obviously does not change the above characteristic, it is also invers to KEV, i.e. "Kill Edge and Face". Any transition in the Eulerian plane can now be represented as a linear combination of the five primitive Euler operators. Each of these or a combination enables the construction of a possible unique topology. They reduce bookkeeping requirements needed to guarantee that the resulting shapes are well-formed, i.e. non-intersecting, closed, and orientable.

We now discuss the conceptual scheme of solids described in boundary representation (see Fig. 3). The structure of a bounded shape model, i.e. OBJECT, is comprised of spatial surfaces named faces. Each FACE is bounded by one or more loops of edges where each loop is the concatenation of line segments, i.e. edges, into a closed RING. EDGES are bounded by VERTICES at their intersections; in our scheme, every edge is given by a START and END vertex, and it topologically points to the LEFT and RIGHT ring respectively.

Fig. 3: Conceptual Scheme of the BR-Approach.

Using a DBMS for storing objects in boundary representation is advantogeous for the following reasons. For instance, the discussed Euler operators are atomic in the sense that they topologically guarantee to handle manifolds consistently, i.e. to fulfill the Euler-Poincaré formula. To construct the topology of a cube for instance, a sequence of a MFVC, seven MEV, and five MEF operators is needed. Thus, the notion of transaction in database theory [Ullman 1982] might be very helpful to better control consistency: The sequence of Euler operators would start with a BEGIN TRANSACTION command and finish with an END TRANSACTION command. The transaction mechanism is such that other transactions (or users) do not see the changes of a transaction until this transaction commits. When it commits, the whole sequence of Euler operators, e.g. the topology of a cube, become visible to other users. If a transaction does not commit but aborts or terminates abnormally, any change to the data are undone and other transactions will see none of the changes.

There are some drawbacks when using relational database technology for bounded surfaces. First, the user is forced to define keys such as O#, F#, R#, E# and V#. These values are necessary to uniquely identify each tuple within a relation. However, to enumerate all faces of an object, all rings of a face, all edges of a ring or both vertices of an edge should be superfluous when interacting through the graphical interface of the solid modeler.

Each instance of the BR-scheme is a group of tuples comprising a single root tuple which defines the object, and several dependent tuples in distinct relations which form its boundary, i.e. faces, rings, edges, and vertices. Even if a *structure* such as PART-OF would be expressed relationally in terms of matching values, it could not be manipulated as a single object. For example, in order to delete an object, the user must issue one delete statement for each tuple in all dependent relations of the object.

Also the *First Normal Form* condition brings disadvantages for a BR-scheme. For instance, a vertex may be described by its positional number and coordinates. Due to normalization, the coordinates may not be treated as vectors but have to be distributed into three attribute domains, namely for x-, y-, and z-values.

3. A SURROGATE MODEL

We now describe a surrogate concept as the basis for an engineering database system, we demonstrate the usability of surrogates for defining PART-OF and IS-A structures in solid modeling, and we give some data retrieval and manipulation considerations.

3.1. Surrogates versus User Keys

In [Hall et al. 1976], it was pointed out that the relational model cannot denote an individual object independently of its attributes. In other words, what would happen if a particular part number (unique identifier) in a CAD database is replaced by a new one: Is it a change to that part number or a replacement by a new part with the same characteristics? To solve this problem, Hall et al. propose to use *surrogates* as "data model representatives" of the entities (unlike tuple identifiers used, e.g. in System R [Astrahan et al. 1976]) and draw the following distinction:

SURROGATES	act as invariant values for individual entities; these values can appear at different places in the database to link entities together.
USER KEYS	act as unique identifiers under user control to identify an individual entity.

One extension of the relational model [Codd 1979] suggests a unary relation for each entity type to list all the surrogates of entities which are currently recorded in the database. Codd's entity integrity constraint allows insertions and deletions of surrogate values but not updates and null values.

Surrogates can be used to provide both fast access and storage independence. Deen's implementation [Deen 1982] employs key compression to provide a more uniform distribution, a hashing algorithm to place tuples on data pages, and an indexing technique to allow fast sequential access. However, Deen's surrogates are similar to tuple identifiers and are generated from primary keys. Therefore, whenever a primary key value changes, its corresponding surrogate also changes.

We introduce a system-controlled attribute SURROGATE and restrict the surrogate values according to the following rules:

- Each SURROGATE value is *system-wide unique*(e.g. concatenation of processor number, database identification, and clock time or sequence number per relation) in order to allow for merging of databases from different sites.

- The values of a SURROGATE attribute *cannot be changed*. The user has no control over the SURROGATE values although they may or may not be made available to him (e.g. it seems appropriate to give surrogate values back to programmers as a program variable).

A SURROGATE acts as an invariant value for each tuple, and no special attribute needs to be chosen as the primary key. In our example of the BR-approach for instance, the user could define O#, F#, R#, E# and V# as surrogates. The system would then automatically generate unique identifiers whenever a tuple is inserted in a relation. In addition, these values could be used to define the structural semantics of the objects.

Very often, the user likes to deal with user-defined primary keys which have some semantic meaning to him. To avoid the introduction of two independent identifier concepts, we introduce a *binding mechanism* between SURROGATES and USER KEYS via a special index called KEY-INDEX. This index is restricted to a single attribute, i.e. unique key, and implies a binding to its corresponding SURROGATE attribute. It is important to note that a user key may or may not exist and may sometimes be changed: Supporting access to an individual tuple of a relation is always guaranteed via the SURROGATE values.

Furthermore, we define two built-in functions to map system-generated SURROGATES onto user defined USER KEYS and vice versa: KEY(surrogate) retrieves the user key corresponding to a surrogate value if one exists, and SURR(user key) retrieves the surrogate value of a specific user key. A one-to-one mapping between internal SURROGATE values and USER KEYS is guaranteed if the attribute of the indexed column is specified with a NOT NULL option. In this case, both functions KEY and SURR yield a unique value which is never null whereas a non-existing operand produces an error message.

3.2. PART-OF and IS-A Structures

Based on the surrogate concept introduced so far, we show how the structural part of our conceptual schemes for solid modeling can be described more directly. In Fig. 2 for instance, the entity set OBJECT can be referred to as root relation which identifies its hierarchical subparts. In order to define this hierarchical structure, we introduce the new attribute SURROGATE for system-generated values in the root relation:

```
RELATION Object;
   ATTRIBUTE
      Art#:            Number;
      O#:              SURROGATE;
      Description:     String20;
      ...
   IDENT
      Art# PRIMARY DOMAIN;
   KEY-INDEX
      Art#.O#;
 END Object;
```

Besides the object number O# (as a surrogate), a user key Art# may be defined and combined with a KEY-INDEX. This index allows the user to retrieve data by article numbers rather than internal surrogates. It also may be used to improve the performance of queries based on the user key attribute, e.g. when searching for tuples with a given article number.

The SURROGATE columns have a semantic meaning besides technical properties such as clustering, avoiding composite keys, and improving performance: They may be used to reference relations. For instance, the dependent relation PART is distinguished by the PART-OF attribute that contains surrogates pointing to tuples in the corresponding parent relation OBJECT:

```
RELATION Part;
   ATTRIBUTE
      P#:              SURROGATE;
      O#:              PART-OF(Object);
      Material:        Classification;
      ...
END Part;
```

Furthermore, an additional column type IS-A may be used to refer to other relations which correspond to a generalization hierarchy. As an example, we consider the relation CYLINDER which is generalized by the relation PRIMITIVE-PART:

```
RELATION Cylinder;
   ATTRIBUTE
      C#:              Number;
      P#:              IS-A(Primitive-Part);
      Radius:          REAL;
      Height:          REAL;
      ...
   IDENT
      C# PRIMARY DOMAIN;
      (Radius,Height) UNIQUE;
END Cylinder;
```

The PART-OF construct is used to define hierarchies of relations and implicitly expresses an existential quantification: For each instance of the hierarchical class, there exist objects constituting its parts. On the other hand, the IS-A hierarchy implicitly expresses a universal quantification: Every instance of a subordinate class has all the properties of the more general class. (Besides PART-OF and IS-A structures, an additional attribute type REFERENCE-OF may be defined to refer to tuples of the same or a different hierarchy. This construct, however, would ask for specific semantics, and performance enhancements would become more difficult, i.e. natural clustering of data may no longer be applicable).

3.3. Data Retrieval and Manipulations

To retrieve data from PART-OF and IS-A hierarchies, a user would often have to join component relations with parent or ancestor relations which requires knowledge of the external structure of a complex object. Instead of defining several join predicates along particular branches involving SURROGATE, PART-OF, and IS-A columns, the user may specify an *implicit join* operator. By this, the whole implicit structure of aggregation or generalization concepts become more transparent and may be easier handled at the user interface.

We discuss the following query based on the relations OBJECT and PART described in the previous section: Show a material list of the article with number 1200.

with implicit join:

```
SELECT  Material
FROM    Object.Part
WHERE   Art#=1200;
```

without implicit join:

```
SELECT  Material
FROM    Object, Part
WHERE   Art#=1200 AND
        Object.O#=Part.O#;
```

The linear implicit join from OBJECT to PART is an equi-join between parent relation and direct child relation. The notation for implicit joins also generalizes to subschemes which are hierarchical rather than linear. Precise definitions and illustrative examples are given in [Meier and Lorie 1983].

An example for using the built-in function KEY is based on the primarily defined KEY-INDEX combining the SURROGATE attribute O# and the USER KEY Art# in relation OBJECT. We consider the following query: Give all article numbers of objects which comprise metal parts.

with KEY-INDEX:

```
SELECT  KEY(O#)
FROM    Part
WHERE   Material=metal;
```

without KEY-INDEX:

```
SELECT  Art#
FROM    Object, Part
WHERE   Material=metal AND
        Object.O#=Part.O#;
```

It should be noted that even if the key value is null, the built-in function KEY can still be performed since only identifiers are needed. Of course, the proposed KEY (and SURR) concept is minimal and helps to avoid writing additional joins to retrieve user keys. It does not help when more than the user key is desired from referenced relations.

The implicit join and the built-in functions KEY and SURR can be used advantageously for insertion and deletion. Although updating through a join is difficult in general, the clean semantics of PART-OF and IS-A structures allow using implicit joins in update statements.

We have discussed a general notion for collecting tuples from different relations by introducing a SURROGATE concept which allows retrieving and manipulating structured data. Since the system knows both the structure and the internal representation of the data from the system catalogs, it can optimize the implicit join accordingly and decide whether or not to use the specialized KEY-INDEX. Accessing solids as interrelated data in a CAD database directly instead of scanning throug different relations improves performance in design work.

4. VECTORS, MATRICES, AND TENSORS

We have argued that in geometric modeling it might be interesting to store the objects with all their necessary geometric and topological information into a database. Whatever representation is chosen for solids, describing points or vectors in coordinate space should be possible. Also, since transformations are common to all geometric modelers, mappings should be supported. The relational model allows only atomic values as attribute elements, therefore it should be augmented [Meloni 1985] to capture *vectors, matrices,* and *tensors.*

A *structured type* of rank m is given by m indices, a dimension vector $n_1,...,n_m$, and each element of that type has $n_1 * ... * n_m$ coordinate values. An index i_k is a sequence of INTEGER values and ranges from 1 up to its corresponding dimension n_k. Each coordinate is an INTEGER or REAL value and may be uniquely identified by a combination of index values out of $i_1,...,i_m$.

Examples of structured types are given in Fig. 4: A structured type of rank 1 with index 1,...,n is a vector of dimension n, a type of rank 2 with indices $1,...,n_1$ and $1,...,n_2$ is a $n_1 * n_2$-matrix and so on.

Fig. 4 Dimension and Rank of Structured Types.

There are two kinds of *operations* for a structured type U of rank r_u with dimension vector $n_1,...,n_{r_u}$ and type V of rank r_v with dimension vector $m_1,...,m_{r_v}$ respectively. The first class of operations leads to a result of unchanged rank and dimension: *Addition* U+V and *subtraction* V-U where the precondition $r_u = r_v$ and $n_i = m_i$ must hold for all i from 1 to $r_u = r_v$; finally, *multiplication* s*U and *division* U/s where s is a scalar. The second class consists of a single operation with a result of indifferent rank and reduced dimension: The *inner product* U*V if the precondition $r_u = r_v$ and $n_{r_u} = m_1$ holds. The new rank is given by the formula $r_{u*v} = r_u + r_v - 2$, and the dimension vector of U*V is built by dropping the last component of the dimension vector of U and the first component of the dimension vector of V and concatenating the rest. As an example, the inner product of a n*m-matrix U and a m*k-matrix V results in the new n*k-matrix U*V.

Since we have introduced a new attribute type, implications for relational operators have to be studied. The traditional mathematical set operations union, intersection, and difference are still possible if the relations are of the same degree, i.e. having the same number of attributes. Also, the Cartesian product for relations with structured types may be defined in the usual way.

The projection operator, however, may now be applied not only to attributes but also to a structured type itself to define a new type of reduced rank or dimension. Therefore, a projection operator becomes a vector, matrix or tensor constructor.

For the selection operator S_F(Relation), we have to generalize the formula F slightly: Constants in a formula may now involve coordinates of a structured type.

The additional operators join and division could be defined by first "unnesting" (compare [Schek and Scholl 1984]) the structured types and applying the usual relational operators. However, performance would become a problem. Also, we don't plan to support a join concept for structured types based on coordinates. We rather restrict join and division for structured types by rank and dimension conditions: Two relations with structured types are called *join compatible* if corresponding attributes show same rank and dimension, and if their values are drawn from the same domain. Of course, structured types involved in a join may first be projected in order to make the relations join compatible.

5. FIRST RESULTS AND CONCLUSIONS

After a decade of research and development activity, relational database systems are now available as products. The flexibility, logical simplicity, and mathematical rigor of such database systems demonstrate a significant new approach to data management, especially in the business application environment. Today, relational database systems are also attracting interest from users outside the commercial areas for which such systems were initially designed. In particular, the need for efficient management of engineering and design data has triggered research on both the requirements of such systems and on extensions to existing database systems.

Some experiments have already been made by extending System R [Lorie et al. 1984] to generate and support surrogates for engineering applications. System R catalogs have been modified to capture the structure of a complex object. The structure information allows the system to analyze the implicit join operator and to find all necessary links in order to materialize the query. Also, a special system table is maintained for each hierarchy of relations to implement a fast intra-object access path. This path is used to enforce parent-child integrity constraints and provides better performance for clustered access and manipulation of tuples which belong to the same complex object.

We have implemented a 3D modeler [Meier et al. 1985] based on a hybrid data structure, namely using a CSG-approach for the user interface but storing the designed objects in boundary representation. Our solids are restricted to plane-faced objects however (see Fig. 5). The user may translate, rotate, or scale objects or may choose a Boolean operation for union, intersection, or difference. Also, a hidden line algorithm is available to better visualize the objects. This modeler POLY has been combined with a relational DBMS developed at our institute in order to study interactions between geometric modeling and databases.

Fig. 5: Using a 3D Modeler for Defining Plane-Shaped Solids.

At present, we are testing a storage structure [Durrer et al. 1985] for complex objects and versions based on the surrogate model. A multiple-tuple-at-the-time interface for PART-OF and IS-A structures is the most important distinction to conventional DBMS. This procedural interface allows us to fetch, copy or delete a complex object by a single database call in order to run geometric and graphical applications more efficiently.

Acknowledgments. I am grateful to Konrad Durrer, Erwin Petry, Erwin Reiner, Walter Reiner, and Andreas Wälchlin for implementing a storage structure based on the surrogate model. Helpful comments on an earlier version of this paper by Klaus Hinrichs and Hans Hinterberger are also acknowledged.

References

[Astrahan et al. 1976]

Astrahan M. M.: System R: Relational Approach to Database Management. ACM Transactions on Data Base Systems, Vol. 1, No.2, June 1976.

[Bieri and Nef 1982]

Bieri H., Nef W.: A Recursive Sweep-Plane Algorithm Determining all Cells of a Finite Division of R^d. Computing, Vol. 28, No. 3, Springer-Verlag 1982, pp. 189-198.

[Codd 1979]

Codd E.F.: Extending the Database Relational Model to Capture More Meaning. ACM Transactions on Data Base Systems. Vol. 4, No. 4, pp. 397-434.

[Deen 1982]

Deen S.M.: An Implementation of Impure Surrogates. Proc. 8th Conf. on Very Large Data Bases, Mexico City 1982, pp. 245-256.

[Durrer 1985]

Durrer K., Meier A., Petry E., Wälchlin A.: Storage Structures for Complex Objects and Versions Based on Surrogates (in German), working paper.

[Eastman and Weiler 1979]

Eastman Ch., Weiler K.: Geometric Modeling Using the Euler Oprators. Proc. First Annual Conference on Computer Graphics in CAD/CAM Systems, MIT 1979, pp. 248-259.

[Hall et al. 1976]

Hall P., Owlett J., Todd S.: Relations and Entities. In: Nijssen G.M. (Ed.): Modelling in Data Base Management Systems. North-Holland, Amsterdam 1976, pp. 201-220.

[Lee and Fu 1983]

Lee Y.C., Fu K.S.: A CSG based DBMS for CAD/CAM and its Supporting Query Language. Proc. of Annual Meeting - Database Week: Engineering Design Applications (IEEE), May 1983, pp. 123-130.

[Lorie et al. 1984]

Lorie R.A., Kim W., McNabb D., Plouffe W., Meier A.: Supporting Complex Objects in a Relational System for Engineering Databases. In: Kim W., Kliner D., Batory D. (Ed.): Query Processing in Database Systems, Springer-Verlag, Berlin 1984.

[Meagher 1982]

Meagher D.: Geometric Modeling Using Octree Encoding. Computer Graphics and Image Processing, Vol. 19, 1982, pp. 129-147.

[Meloni 1985]

Meloni T.: Extension of the Relational Algebra by Vectors, Matrices, and Tensors (in German). Semesterarbeit, Informatik, ETH Zürich, 1985.

[Meier and Lorie 1983]

Meier A., Lorie R.A.: Implicit Hierarchical Joins for Complex Objects. IBM Research Report RJ3775, San Jose 1983, pp. 1-13.

[Meier et al. 1985]

Meier A., Paquet F., Petry E.: POLY - A Solid Modeler on the LILITH Workstation (in German). Benutzeranleitung, Informatik, ETH Zürich, 1985.

[Requicha 1980]

Requicha A.A.G: Representations for Rigid Solids: Theory, Methods and Systems. Computing Surveys, Vol. 12, No. 4, December 1980, pp. 437-464.

[Schek and Scholl 1984]

Schek H.-J., Scholl M.H.: An Algebra for the Relational Model with Relation-Valued Attributes. Technical Report DVSI-1984-T1, TU Darmstadt, 1984.

[Tilove 1980]

Tilove R.B.: Set Membership Classification: A Unified Approach to Geometric Intersection Problems. IEEE Transactions on Computers, Vol. C-29, No. 10, October 1980, pp. 874-883.

[Ullman 1982]

Ullman J.D.: Principles of Database Systems. Computer Science Press, 1982.

Figures

Repres. Scheme	Primitive Instancing	Spatial Enumeration	Cell Decomposition	Constructive Solid Geomtry	Boundary Representation
Database Technique	applicable	not adequate	not adequate	applicable	applicable

Fig. 1: Using Database Techniques for Solid Modeling.

```
    D ← φ
Endwhile
```

Since the segments are deleted in order of increasing height, when a segment is deleted its adjacent segments in W are the two nearest segments on either side which are at least as tall. To guarantee this property, it is not necessary to delete the segments in strict a_i order. As long as each segment deleted is currently a local minimum in W this property will hold. The following algorithm determines the visibilities *without the sorting of the a_i's* in linear time by scanning W from left to right and always deleting the left most local minimum.

To avoid checking the boundary conditions while traversing W, two segments s_0, s_{n+1} are added:

s_0 = [illegible] ∞ [illegible], $x_0 < x_i$ for all i, $1 \le i \le n$

s_{n+1} = [illegible] ∞ [illegible], $x_i < x_{n+1}$ for all i, $1 \le i \le n$

```
    [illegible]
    [illegible]
        [illegible]
            ← [illegible]
        [illegible]
         ← [illegible]
        [illegible]
            Report [illegible]
            [illegible] ← [illegible]
             ← [illegible]
        Endwhile
        Report (i,n(j))
        n(j) ← n(i),  (n'(i)) ← j
        i ← j
    Endwhile
```

x-order, i.e., $x_i \le x_{i+1}$ for all i. Further we assume without loss of generality that $x_i < x_{i+1}$ for all i. For both cases, we will first propose linear time algorithms, assuming the s_i's are also in sorted a_i- and b_i- order. Later on, we will show how to remove this assumption without affecting the time complexity.

(1) *The rooted case*

Segments $s_1, s_2, \ldots, s_n$ are given such that $s_i = (x_i, a_i, b_i)$ and $a_i \ge 0, b_i = 0$ for all i. Also assume the s_i's are in x_i-sorted order.

In this case if s_i and s_j see each other, then they see each other at the top of the shorter of the two segments. Thus by reporting what each segment sees at its top, all visibilities will be generated.

If the sorting of the a_i's is available, then the visibilities can be obtained by scanning in the increasing vertical direction, using a horizontal line. During this scan the list in x-order of segments with a_i above the line is kept. As each a_i is encountered, s_i is deleted and the adjacent segments on either side are reported as visible from s_i. If segments of the same length are allowed, then care must be taken to report and delete these segments, simultaneously.

```
W = List of remaining segments in x-order. n(i) and p(i) denote the segments next and prior,
respectively, to segment s_i in W
D = List of segments to be deleted, simultaneously
    = sorting of s_i's by a_i: $a_{i_k} \le a_{i_{k+1}}$ for all k

D ← ∅, k ← 1
Do while k ≤ n
        L ← $a_{i_k}$
        Do while $a_{i_k}$ = L and k ≤ n
                Report $(i_k, n(i_k))$, Report $(p(i_k), i_k)$
                D ← $i_k$, k ← k + 1
        Endwhile
        Delete all elements of D from W
```

2 Determine the strip $y_{min} = \max(b_i, b_j), y_{max} = \min(a_i, a_j)$ of possible visibility for s_i, s_j (If $y_{min} > y_{max}$, then stop with answer "no", that is, s_i, s_j are not visible). Cut out the portions of segments s_{t_u} that fall outside of this strip, namely, change each s_{t_u} to s'_{t_u} such that $b'_{t_u} = \max(b_{t_u}, y_{min})$, $a'_{t_u} = \min(a_{t_u}, y_{max})$. Eliminate all segments with $b'_{t_u} > a'_{t_u}$, thus obtaining the new sequence $s_{t_1}, \ldots, s_{t_h}$, $h \leq k$. (If this sequence is void, then stop with answer "yes".)

3 If $b_{t_1} > y_{min}$, then stop with answer "yes". Otherwise, for u ranging from 1 to h, build

$$\bigcup_{v=1}^{u}[a_{t_v}, b_{t_v}]$$

by adding a new interval $[a_{t_u}, b_{t_u}]$ one at a time. If the interval union breaks up, that is, b_{t_u} is greater than the upper extreme $\bar{a}$ of

$$\bigcup_{v=1}^{u-1}[a_{t_v}, b_{t_v}],$$

then stop with answer "yes".

4 If the upper extreme $\hat{a}$ of $\bigcup_{v=1}^{h}[a_{t_v}, b_{t_v}]$ is less than y_{max}, then stop with answer "yes", else stop with answer "no".

Steps 1, 2 and 3 require $O(n)$ time, and step 4 requires constant time. It may be noted that, if no sorting in the a_i-, b_i- order is assumed, visibility of s_i, s_j can be solved in $O(n + h \log n)$ time, where $h \leq n-2$ is the parameter found in step 2 above. In fact, the elements of the sequence $s_{t_1}, \ldots, s_{t_h}$ can be found with two linear scans on the x- and y-coordinates. Sorting this sequence is however needed to determine visibility.

3 Special cases

In this section we study several special cases of the visibility problem. Linear time algorithms are proposed even when the s_i's are not sorted in the a_i- and b_i- order. The first case is the *rooted case*, where $b_i = 0$, $a_i \geq 0$, $i = 1, 2, \ldots, n$. The second case is the *double comb case*, where $0 \leq a_i \leq b_i \leq A$, and for each i, $a_i = A$ or $b_i = 0$. We assume the s_i's are in sorted

Lemma 5. Determining whether two arbitrary segments s_i, s_j from a set of input segments form a visibility pair requires $\Omega(n \log n)$ steps if the s_i's are not sorted in the a_i - and b_i - order.

Proof. First we prove that, if an algorithms $\mathscr{A}$ of order $< O(n \log n)$ existed for this problem, then the problem of determining whether

$$\bigcup_{i=1}^{n} [a_i, b_i]$$

is an interval (or splits into several disjoint intervals) could also be solved in the same time. In fact, given a set of n intervals $[a_i, b_i]$, assign to each of them a random coordinate value x_i, and interpret them as vertical segments s_i. Then compute the maximum value y_{max} of all a_i, the minimum value y_{min} of all b_i, and the minimum and maximum values x_{min}, x_{max} of all x_i. All this can be done in $O(n)$ time. Letting $x_0 = x_{min} - 1$, $x_{n+1} = x_{max} + 1$, create two new segments s_0, s_{n+1} represented by the triples (x_0, y_{max}, y_{min}), $(x_{n+1}, y_{max}, y_{min})$, and consider the set $S = \{s_0, s_1, \ldots, s_n, s_{n+1}\}$. s_0, s_{n+1} form a visibility pair if and only if

$$\bigcup_{i=1}^{n} [a_i, b_i]$$

is not an interval. Algorithm $\mathscr{A}$ could be applied to S to determine visibility between s_0 and s_{n+1}, thus solving the union problem for $[a_i, b_i]$. However, this latter problem requires $\Omega(n \log n)$ steps, as shown in Appendix 1. We conclude that $\mathscr{A}$ cannot exist.

□

Let us now examine if sorting of the s_i's in the x_i-, a_i- or b_i- order enables us to solve the visibility problem of s_i and s_j in less than $O(n \log n)$ time. Sorting in the x_i - order would not help, because the only thing relevant is to find the segments whose x-coordinates fall in the interval $[x_i, x_j]$, which can always be determined in linear time. Sorting in the a_i - or b_i - order, on the other hand, makes the problem solvable in time $O(n)$, if we apply the following procedure (assume $x_i < x_j$, and the s_i's are in sorted b_i - order).

1. Scan the sequence of all segments in order of increasing values of b_i, and extract the ordered subsequence $s_{i_1}, \ldots, s_{i_k}$ of those segments with $x_i < x_{i_u} < x_j$, $1 \le u \le k$.

Proof. Consider the example in Fig. 4. The visibility pairs are:

(1, n), (1,2), (1,3), . . , (1, n-1),

(2, n), (2,3), (3,n), (3,4), . . ,

(n-1, n),

with a total of (n-1) + 2(n-3) + 1 = 3n-6.

□

We present, without proof, the following remark as a Corollary to Lemma 3.

Definition. Given a set of segments $S = \{s_1, s_2, \ldots, s_n\}$, we define a *visibility graph* G_S as follows: S is the set of nodes for G_S. There is an edge between two nodes s_i and s_j if and only if (i,j) is a visibility pair.

Remark. Visibility graphs are planar graphs.

We now prove a lower bound for the visibility problem, assuming the s_i's are not sorted in the x_i - order.

Lemma 4. The visibility problem requires Ω (n log n) time if the s_i's are not sorted in the x_i - order.

Proof. Given a set of n integers $W = \{w_1, w_2, \ldots, w_n\}$, for each i construct a segment $s_i = (x_i, [illegible])$, where $x_i = w_i$. Solving the visibility problem implies finding the two neighbors of each segment, i.e. the one before and the one after on the x-axis. In other words, for each w_i we know the one just smaller than and the one just larger than w_i. From this, we can construct a sorted list of W in linear time.

□

We next prove a lower bound for the visibility problem under the assumption that the s_i's are sorted in the x_i - order but not in the a_i - or b_i - order. We will first prove a stronger result which involves only two arbitrary segments.

Lemma 1 If [illegible] is a visibility pair then

(i) the line y=min[illegible] intersects s and s without intersecting any s_k with $i<k<j$

or (ii) there exists m [illegible] such that [illegible] and for every [illegible], the line y= [illegible] intersects s and s without intersecting any s_k with [illegible]

Proof If [illegible] and [illegible] are a visibility pair then there [illegible] intersects [illegible] and [illegible] for [illegible] Move this line $y=d$ upwards. Then either [illegible] or else some segment s [illegible] will be encountered. The former case corresponds to (i) while the latter case [illegible] and [illegible]

□

In the following we say that [illegible] reports [illegible] or [illegible] reports [illegible] if case [illegible] of Lemma 1 [illegible] that [illegible] case (ii) of Lemma 1 occurs

Lemma 2 The number of [illegible] visibility pairs for n segments is at most 3n-6, for $n \geq 3$

Proof Let S = [illegible] given segments and v the number of visibility pairs in S [illegible] Define [illegible] b_n [illegible] S' [illegible] Denote by v' the number of visibility pairs in S' [illegible] that $v \leq v'$ [illegible] that $v' \leq 3n-6$

[illegible] visibility pair in S'. If [illegible] then [illegible] will be reported by [illegible] some [illegible] with [illegible] Lemma 1 (See Fig. 2) If [illegible] then [illegible] will be reported [illegible] If [illegible] then [illegible] will be reported [illegible] then define b^*=min [illegible] the visibility pair will be reported by b^*. It is concluded that every visibility pair in S' is reported by some [illegible] with $2 \leq i \leq n-$ [illegible] or by some b_m with $2 \leq m \leq n-1$. Further, every [illegible] $(2 \leq i \leq n-1)$ may report at most two [illegible] (namely some [illegible] with [illegible] and some [illegible] with [illegible] while any b_m $(2 \leq m \leq n-$ [illegible] may report at most one visibility pair (namely some [illegible] with [illegible] $< m < k$). We thus have $v' \leq 2(n-2)+n-2 = 3n-6$

□

Lemma 3 3n-6 is a tight upper bound

In the second part of the paper, we first assume that the s_i's are in sorted x_i-order but not in sorted a- or b-order. We prove that

(1) When all a_i, $i=1,2,\ldots,n$ are the same, $O(n)$ time suffices to solve the visibility problem.

(2) If adjacent segments in x-order have overlapping y-projections, $O(n)$ time suffices.

(3) Given A, such that $1 \le A$ [illegible] $\le n \le$ [illegible] $\le A$ and for [illegible] $1 \le i \le n$, either [illegible] $= A$ or [illegible] $=$ [illegible] holds, then the visibility problem can be solved in $O(n)$ time.

Linear time algorithms are presented for the abovementioned special cases.

Finally, we assume that the s_i's are in sorted x-, a- and b-order. We propose a strategy for the general problem which divides some of the segments into smaller ones. Let N be the total number of these segments. The algorithm will run in $O(N)$ time. In some special cases $N=O(n)$, for example, when all segments are of equal length or when the y-axis projection of no segment is contained completely in that of any other one, this is true. But in general N can be as large as $O(n^2)$. Thus, before running the algorithm itself one may want to spend $O(n)$ time in preprocessing to find out the exact value of N.

Moreover, there is a straightforward algorithm which solves the visibility problem in $O(n \log n)$ time, which will be described at the end of the paper.

Complexity of the Visibility Problem

In this section we will present some results on the complexity of the visibility problem, given whether the input is sorted or not. First, we show that given n segments, $n \ge 3$, the number of visibility pairs is at most $3n-6$. Thus the output size is bounded above by $3n-6$. Consequently, we can't [illegible] a proof [illegible] preclude the existence of linear time algorithms for the visibility problem. We then show that for every n, the bound $3n-6$ is indeed achievable.

Throughout this paper, we will also use (i,j) to denote the visibility pair (s_i, s_j) for $1 \le i < j \le n$ and $i < j$ to denote that s_i lies to the left of s_j, i.e., $x_i < x_j$. We need the following lemmas.

must exceed a given number. In general, spacing constraints are assigned only to pairs of elements which can "see" each other (visibility pairs). Again, in Fig. 2, element pairs (1,3), (1,2), (2,3) are such pairs.

Even though the elements are geometrical shapes consisting of horizontal and vertical edges, as far as determining visibility pairs is concerned, it suffices to represent them by their center lines, which will be referred to as vertical line segments. See Fig. 1 (b) for illustration. We can now formulate the visibility problem precisely.

Let $s_1, \ldots, s_n$ be n vertical segments in the plane. (A segment is assumed to include its two endpoints.) Two segments are visible to each other if there exists a horizontal line intersecting them which does not intersect any other vertical segment between them. Such segments are [illegible] determine all visibility pairs efficiently.

Each segment s_i is represented by a triple (x_i, a_i, b_i) where x_i is its x-coordinate, a_i its top [illegible] y-coordinate. In general, we can assume that the s_i's are sorted [illegible] by the editor in the graphic system before entering the compaction [illegible] has to determine the visibility pairs. This is because the graphic system can do the sorting in an on-line fashion as the designer enters the elements into the system. But visibility pairs can be computed only after all the segments are positioned and is in general performed by the compactor.

In this paper we [illegible] problem under various assumptions. Specifically, we [illegible] that the number of visibility pairs is $O(n)$, where n is the number of vertical segments. Thus the output size will not preclude the existence of linear-time algorithms as in most cases in computational geometry. Secondly, we show that if the s_i's are [illegible] then $\Omega(n \log n)$ time is necessary to determine all visibility pairs. On the other hand, if the s_i's are in sorted x_i-order, but not in sorted a_i- and b_i-order, then $\Omega(n \log n)$ time is still necessary. In fact, we can prove a stronger result, namely, if the s_i's are not sorted in both the a_i- and b_i-order, then determining the visibility of any given pair of segments (s_i, s_j) requires $\Omega(n \log n)$ time (in this case, whether the s_i's are in sorted x_i-order or not is not essential).

1 Introduction

A symbolic layout system is an automated design tool for laying out the masks of a VLSI circuit. In this system, circuit elements such as transistors, contacts, capacitors, etc. which are composed of overlapped mask levels, are represented by their respective graphic symbols. Interconnecting mask polygons used as wires are also symbolically represented by their center lines. When a designer places the symbol of a circuit element on the layout plane, it signifies that all the mask levels composing the element exist there. This type of symbolic layout is also called a stick diagram [1].

After a stick diagram is drawn through a graphic system, a compaction system will automatically space the circuit elements and interconnections to obtain a mask physical layout. The compactor considers the connecting wires as stretchable and packs the circuit elements as [illegible] possible without violating the spacing constraints defined by the user as well as by [illegible] [2-7]. If the compactor is sufficiently efficient, the designer can afford to [illegible] the circuit's topology until a satisfactory geometrical layout is arrived.

To simplify the design of the compaction algorithm, the system requires that only [illegible] structures be allowed. Thus, the edges of the geometrical shape of every mask [illegible] are either in the horizontal (X) or in the vertical (Y) direction. Consequently, all the [illegible] constraints are also in either of these directions. Because of lack of better algorithms, the [illegible] system first compacts the layout optimally in one direction and then similarly [illegible] in the other direction. Thus, it suffices to consider the compaction problem [illegible] in the X direction.

[illegible] the compaction algorithm must be very efficient and [illegible]. Before the compaction algorithm can even begin, spacing constraints must be [illegible] to the relevant elements. How to efficiently determine the relevant elements is the subject of this paper.

Because the elements and geometrical shapes consist of horizontal and vertical edges (Fig. 1 (a)), the distance between two elements is defined as the minimum distance between corresponding vertical edges of the elements. For example, in Fig. 2, the distance between elements 1 and 3 is [illegible]. By the spacing constraint on two elements we mean their distance

A VISIBILITY PROBLEM IN VLSI LAYOUT COMPACTION

M. Schlag[1,5], F. Luccio[2,5], P. Maestrini[2,5], D. T. Lee[3,5], and C. K. Wong[4]

1 Department of Computer Science, University of California, Los Angeles, CA 90024
2 Istituto di Scienze dell'Informazione, Universita di Pisa, Corso Italia, 40-56100 Pisa, Italy
3 Department of Electrical Engineering and Computer Science, Northwestern University, Evanston, IL 60201. This author's research was supported in part by the National Science Foundation under Grants MCS-8202359 and ECS-8121741
4 IBM T. J. Watson Research Center, P. O. Box 218, Yorktown Heights, NY 10598
5 This work was performed while this author was visiting IBM T. J. Watson Research Center

Abstract: Given a set of n vertical segments $s_1, s_2, \ldots, s_n$, two segments are visible to each other if there is a horizontal line intersecting them which does not intersect any other vertical segment between them. We study the problem of determining all visibility pairs. Each segment is represented by a triple (x, a, b), where x is the x-coordinate of segments s, a its top y-coordinate and b its bottom y-coordinate. Various complexity results are given, according as the s's are in sorted x-, a-, or b-order. O(n) time algorithm is presented for various special cases. A dividing strategy is proposed for the general problem with time complexity O(N), where N is the number of final segments. Alternately, a very simple O(n log n) algorithm can also be used to solve the general problem. Motivation and application in VLSI layout compaction is also discussed.

in: F. P. Preparata, Ed.
Advances in Computing Research, vol. 2
JAI Press 1985, 259-282

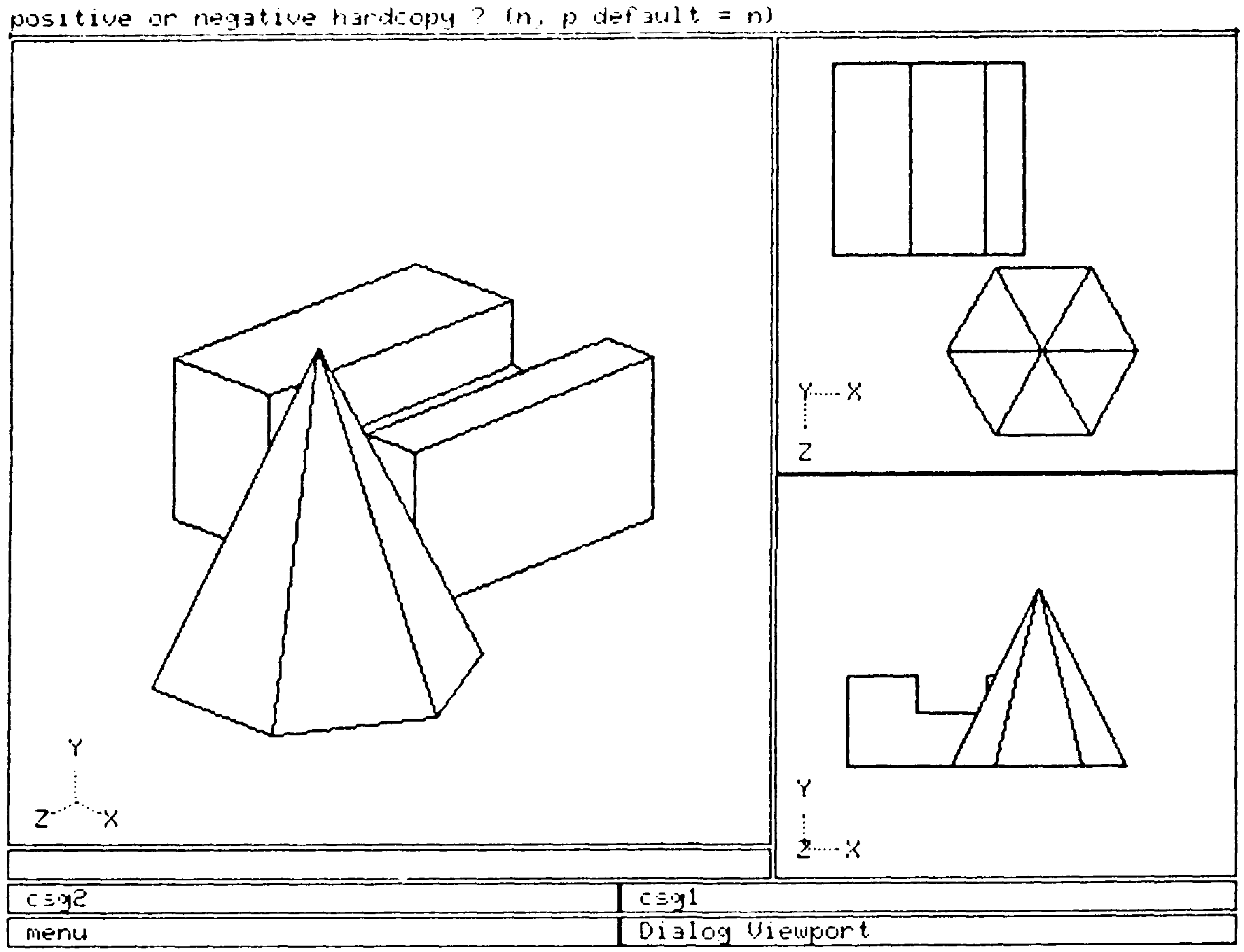

positive or negative hardcopy ? (n, p default = n)

Fig. 5: Using a 3D Modeler for Defining Plane Shaped Solids.

	STRUCTURED TYPE			
	SCALAR	VECTOR	MATRIX	TENSOR
DIMENSION	0	n	n_1, n_2	$n_1, \ldots, n_m$
RANK	0	1	2	m

Fig. 4 Dimension and Rank of Structured Types.

OBJECT

O#	DESCRIPTION

FACE

F#	FACE-O#	COLOR

RING

R#	RING-F#

EDGE

E#	START-V#	END-V#	LEFT-R#	RIGHT-R#

VERTEX

V#	X-COORDINATE	Y-COORDINATE	Z-COORDINATE

Fig. 3: Conceptual Scheme of the BR-Approach.

OBJECT

O#	DESCRIPTION

PART

P#	PART-O#	MATERIAL	GENERIC-TYPE

TRANSFORMED-PART

P#	MOVED-P#	PARAM1	PARAM2	PARAM3	MOTION

PRIMITIVE-PART

P#	X-LOCATION	Y-LOCATION	Z-LOCATION	GENERIC-TYPE

CUBE

CONE

CYLINDER

SPHERE

COMBINED-PART

P#	FIRST-P#	SECOND-P#	OPERATION

Fig. 2: Conceptual Scheme of the CSG-Approach.

Note that at any time during the algorithm, all the segments to the left of s_i are at least as tall as s_i.

Suppose $i_1 < i_2 < \dots < i_m$ for some $m \geq 1$ are deleted during one execution of the interior of the outer while loop. Then,

1) $a_{p(i_1)} > a_{i_1} = a_{i_2} = a_{i_3} = \dots = a_{i_{m-1}} = a_{i_m} < a_{n(i_m)}$
2) $n(i_k) = i_{k+1}$ for all k, $1 \leq k \leq m-1$
3) $p(i_k) = i_{k-1}$ for all k, $2 \leq k \leq m$

Lemma 6 Assume $i<j$. Then i and j see each other if and only if i and j become adjacent in W.

Proof (Sufficiency ⇒)

Suppose i sees j. Then $a_k < \min(a_i, a_j)$ for all k, $i<k<j$. Suppose i and j do not become adjacent in W. Then there must be some segment between them, s_k, $i<k<j$ which is not deleted before either i or j.

If i and j are deleted at the same time, then by 1) $a_i = a_j$, and by 2) and 3) k must also be deleted. However then $a_k = a_i = a_j$, which is not possible.

If i and j are not deleted at the same time, then let k be the segment adjacent to i or j, whichever one is deleted first, at the time of this deletion. Then by 1) $a_k \geq$ the height of this segment, so $a_k \geq \min(a_i, a_j)$ which is not possible.

(Necessity ⇐)
Assume i and j become adjacent in W, but do not see each other. Then there exists k, $i<k<j$ such that $a_k \geq \min(a_i, a_j)$.

Case 1. $a_i \leq a_j$

Let k be the closest segment to i, such that $i<k<j$ and $a_k \geq a_i$. Such a k must exist if i does not see j. In this case all segments between i and k are shorter than i and k, so k will not be deleted until it becomes adjacent to i. But then since $a_k \geq a_i$, k will be deleted at the same time or after i, and i will never become adjacent to j.

Case 2. $a_i > a_j$.

Let k be closest to j such that $i<k<j$ and $a_k \geq a_j$. Then by the same reasoning, k cannot be deleted before i and so i and j cannot become adjacent to i in W.

$\square$

The correctness of the algorithm follows directly from the lemma, since each segment is deleted from W and at that time the segments it sees at its top will be reported.

An example to illustrate the algorithm is given in Fig. 5, where $s_1 = (1,0,4)$, $s_2 = (2,0,5)$, $s_3 = (3,0,6)$, $s_4 = (4,0,3)$, $s_5 = (5,0,2)$, $s_6 = (6,0,1)$, $s_7 = (7,0,2)$ and $s_8 = (8,0,4)$. The detailed execution of the algorithm is also described.

(2) *The overlapping case*

Assume any two adjacent segments s_i, s_{i+1} have overlapping projections on the y-axis for every i. Let $L = \min\{b_i, 1 \leq i \leq n\}$ and $U = \max\{a_i, 1 \leq i \leq n\}$. (See Fig. 6). We consider the following two rooted cases:

(a) Extend all the original segments downwards to reach L.

(b) Extend all the original segments upwards to reach U.

Solve both cases (a) and (b). The combined result will include all visibility pairs (possibly with duplication). This is because of the following property. Consider any two segments s_i, s_j which see each other and consider any horizontal line K intersecting them which does not intersect any other segments between s_i and s_j. Then either no segments are between s_i, s_j or they must be *all* above K or *all* below K. In either case, the visibility pair (i,j) will be reported in either Case (a) or Case (b), or both.

(3) *The double-end case*

The input segments $s_1, s_2, \ldots, s_n$, $s_i = (x_i, a_i, b_i)$ *for all i, are such that* $0 \leq a_i \leq b_i \leq A$ and for each i, $a_i = A$ or $b_i = 0$. The s_i's are in sorted x_i-order. (See Fig. 7).

The segments are assumed to be separated into two lists, T and B, which are doubly linked using n(i)'s and l(i)'s in increasing x_i-order. T_0 is the initial element of T, B_0 is the initial element of B. T consists of those segments with $a_i = A$ and B consists of those segments with $b_i = 0$. Note that a segment with $a_i = A$ and $b_i = 0$ can be in either list, but will be assumed to be in only one.

The algorithm for this case is divided into two phases. During the first phase the visibilities among segments of T will be obtained. This is accomplished as in the rooted case, by scanning from left to right the list of the T segments and deleting the left local maximum, thus processing the T - segments as an upside down rooted case. However, reporting visibilities will be different since the B segments may affect the visibility of T segments.

The following are necessary and sufficient conditions for two T-segments to see each other (s_i and s_j), $i<j$.

1) $j = next(i)$ initially and the tallest B-segment, s_k with $i<k<j$ has $a_k < A$.

2) During processing of the list i and j become adjacent after the removal of m, $i<m<j$ and the tallest B-segment, s_k with $i<k<j$ has $a_k < b_m$.

To check visibility of T-segments during the algorithm, the height of the tallest B-segment with k, $i<k<next(i)$ is kept for each i. RB(i) will denote the height of this B-segment.

Additional information is retained for use in the second phase. This information consists of which T-segment a T-segment sees at its tip (if any). This information is stored in the arrays VTR and VTL.

Assume two boundary segments s_1 and s_{n+1} have been added to the T list. Define n of the last B-segment as n+2.

Phase One

i is the pointer for the T-list.

j is the pointer for the B-list.

Preprocess

$i \leftarrow 0;\ j \leftarrow B_0$

```
Do while $i \neq n+1$
        $M \leftarrow 0$
        Do while $j < n(i)$
                $M \leftarrow \max(a_j, M)$
                $j \leftarrow n(j)$
        Endwhile
        $RB(i) \leftarrow M$
        If $M < A$ then Report $(i, n(i))$ fi
        $i \leftarrow n(i)$
Endwhile
```

Process

```
$C \leftarrow$ number of I-segments (excluding $s_0$ and $s_{q+1}$)
            ($j$ is no longer a pointer to B-list here)
$i \leftarrow 0$
Do while $C > 0$
        Do while $b_{[illegible]} \geq b_{[illegible]}$
                $i \leftarrow n(i)$
        Endwhile
        $j \leftarrow i$ ; $M \leftarrow 0$
        Do while $b_{[illegible]} = b_{[illegible]}$
                If $RB(j) < b_{[illegible]}$ then $VTR(j) \leftarrow n(j)$ fi
                If $RB([illegible](j)) < b_{[illegible]}$ then $VTL(j) \leftarrow [illegible](j)$ fi
                $M \leftarrow \max(M, RB(j))$; $C \leftarrow C-1$; $j \leftarrow [illegible](j)$
        Endwhile
        If $M > RB(j)$ then $RB(j) \leftarrow M$ fi
        If $RB(j) < b_{[illegible]}$ then Report $(j, n(i))$ fi
        $[illegible](n(i)) \leftarrow j$ ; $n(j) \leftarrow n(i)$
        $i \leftarrow j$
Endwhile
```

If the sorted orders of the a_j's and b_i's are known then the second phase can be accomplished using the same vertical scan technique as for the rooted case. A list L is kept of those segments, from left to right, which intersect the horizontal scan line. Initially all B-segments are in the list L. As the scan line is moved up, B-segments are deleted from L and T-segments are inserted into L. The difference with the rooted case algorithm is that the T-segments must be inserted in constant time while preserving the x-ordering of the list. To accomplish this task, the VTL's and VTR's obtained during phase one are used. If VTL(i)≠0 (i.e. defined) then i can be inserted in the list L to the right of VTL(i). If VTR(i)≠0 then i can be inserted in the list L to the left of VTR(i). This is because VTR(i) and VTL(i) are always segments of the same or greater length. Care must be taken however with segments of equal length. It will be assumed in the ordering of the b's of the T-segments that if b_k and b_j are equal then b_k appears after b_j if k is to the left of j, i.e., if the T-segments are to be scanned in increasing [illegible] order the right one (in the x order) precedes the left one if their b's are equal. The same assumption will be made for B-segments with equal a's.

Thus if VTR(i) is defined for i, a T-segment, then VTR(i) will have already been inserted into the list. There are, however, T-segments for which VTL(i)=VTR(i)=0 (not defined), or VTR(i)=0 and VTL(i) is a segment of the same length. In either case we shall use auxiliary pointers AP determined during preprocessing to insert i.

Recall that VTR(i) is undefined only when there exists a B-segment j that blocks the T-segment i from seeing any T-segment on its right, namely, $a_j \geq b_i$ for some j and i<j. Therefore we shall scan the lists of B- and T-segments from left to right and compute the auxiliary pointers AP for each T-segment. Note that if VTR(i) is defined then AP(i) is set equal to VTR(i), which is a T-segment and if VTR(i) is undefined then AP(i) will be equal to the first B-segment j (i<j) such that $a_j \geq b_i$. Because of our assumptions above and the assumption that if $b_k = a_j$ for some T-segment k and B-segment j, then k precedes j, when a T-segment i is encountered, AP(i) must be present in the list L and hence segment i can be inserted to the left of AP(i).

Suppose that segment i_1 is the first T-segment for which VTR(i_1) is not defined and that before a B-segment j with $a_j \geq b_{i_1}$ is encountered in the left-to-right scan we have seen segments $i_2, \ldots, i_k$ all of whose VTR's are undefined. From the definition of VTR's we know that $b_{i_1}<b_{i_2}<\dots<b_{i_k}$. So when segment j is seen, we simply set AP(i) to j for all i in $\{i_1,i_2,\dots,i_k\}$ such that $b_i<a_j$. This is done by scanning the list $b_{i_1},b_{i_2},\dots,b_{i_k}$ in that order until $b_{i_m}>a_j$, $1\le m\le k$, is satisfied or the list is exhausted. Thus, at any moment we have a list UL of T-segments whose VTR's are undefined and whose b's are in strictly increasing order. When a B-segment j is seen, we simply compare its a-value with the b-value of the first segment in the list UL. If it is greater, then AP(i) is set to j and i is deleted from UL; the same process repeats. Otherwise, we discard j and continue scanning. During the scanning any T-segment with defined VTR value is ignored (except that the value is assigned to the corresponding AP field).

Once the auxiliary pointers are found, the insertion of a T-segment during the vertical scan can be done in constant time.

Visibilities will be reported as follows:

Among B-segments

1. When a B-segment is deleted, then if its neighbors are B-segments, it will report seeing them.
2. When a T-segment is inserted between two B-segments, then the visibility of these two B-segments will be reported.

Between T- and B-segments

1b. When a T-segment is inserted, if its neighbor is a B-segment then it will report seeing it.

2b. When a B-segment is deleted, if its neighbors are not both T- or both B-segments, then their visibility is reported.

We make the remark here that because there may be segments of equal length, in case 1b the visibility of a T-segment i and its left neighboring B-segment j is reported only when VTR(i) is not defined and in case 2b the visibility of T-segment i and B-segment j is reported when $a_j > b_i$.

Phase Two

For this algorithm it is assumed that $i_1, i_2, \ldots, i_n$ is in the ordering of B- and T-segments by a_i and b_i collectively in increasing order such that

1) If s_{i_k} and s_{i_m} are both T-segments with $b_{i_k} = b_{i_m}$ and s_{i_k} is to the right of s_{i_m}, then $k<m$

2) If s_{i_k} and s_{i_m} are both B-segments with $a_{i_k} = a_{i_m}$ and s_{i_k} is to the right of s_{i_m}, then $k<m$

3) If s_{i_k} is a B-segment and s_{i_m} is a T-segment such that $a_{i_k} = b_{i_k}$, then $m<k$, i.e., a T-segment precedes a B-segment in the ordering when their a- and b-values are equal

Preprocess

```
    $k \leftarrow 1$
    APL ← t  (APL is a queue and is initialized to contain a segment t whose b-value is
          $A+1$ and APL ⇐ i and j ⇐ APL are insertion and deletion operations on APL
          respectively.)
    Do while $k<n$
       $k \leftarrow k+1$
       Do while $k<n$ and Type$(i_k)$ = T and VTR$(i_k) \neq 0$
            $k \leftarrow k+1$
       Endwhile
       If Type$(i_k)$ = T then APL ⇐ $i_k$
          else if Type$(i_k)$ = B then j ⇐ APL;
                        Do while $a_{i_k} \geq b_j$
                           AP(j) ← $i_k$
                           j ⇐ APL
                        Endwhile;
                        APL ⇐ j fi
       fi
       If $k = n$ then j ⇐ APL;
                  Do while $b_j \neq A+1$
                     AP(j) ← 0
                     j ⇐ APL
                  Endwhile
```

```
        fI
    Endwhile
```

{AP(j) is the position that is immediately to the right of the T-segment j when j is inserted in the list L below. If AP(j)=0 then j is to be inserted into the rightmost position of L.}

Initialize the list L to contain the B-segments in increasing x_1-order and maintain it as a doubly-linked list

```
k ← 1
Do while k ≤ [illegible]
        If Type($i_k$)=T then Value ← $b_{i_k}$ else Value ← $a_{i_k}$ fI
        Do while Type($i_k$)=T and $b_{i_k}$ = Value
            If AP($i_k$)≠0 then Insert $i_k$ to the left of AP($i_k$)
                                  else Insert $i_k$ at the end of L
            fI
            If Type(n($i_k$))=B then Report ($i_k$,n($i_k$)) fI
            If Type($\ell$($i_k$))=B and VTI ($i_k$)=0 then Report ($\ell$($i_k$),$i_k$) fI
            If Type(n($i_k$))=B and Type($\ell$($i_k$))=B
                then Report ($\ell$($i_k$), n($i_k$)) fI
            k ← k+1
        Endwhile
        Do while Type($i_k$)=B and $a_{i_k}$ = Value
            If Type(n($i_k$))=B then Report ($i_k$,n($i_k$)) fI
            If Type($\ell$($i_k$))=B then Report ($\ell$($i_k$),$i_k$) fI
            If Type(n($i_k$)) ≠ Type($\ell$($i_k$)) and $a_{i_k}$ < min ($a_{\ell(i_k)}$,$a_{n(i_k)}$)
                then Report ($\ell$($i_k$),n($i_k$)) fI
            Delete $i_k$ from L
            k ← k+1
        Endwhile
    Endwhile
```

Double Comb without a_i or b_i sortings

For a segment, s_i, the set of visible segments, { j | (i,j) or (j,i) is a visibility pair } is simply the set of nodes which become adjacent to node i in the list sometime during the scan. It is possible to determine these segments without actually performing the scan, and thus without using the a_i or b_i sortings.

Let RV(i) = { j | (i,j) is a visibility pair and j>i }. RV(i) is the set of segments on the right of s_i which are visible to s_i. Clearly, if RV(i) is determined for each s_i then all visibility pairs will be reported. Suppose now that s_i is a B-segment and consider the following two sequences of segments, B(i) and T(i).

B(i) = $\{s_{q_j}\}_{j=1}^{k}$ such that s_{q_j} is a B-segment with $i<q_j$ and $a_{q_j}<a_{q_{j+1}}$ for all $1\leq j\leq k$. In addition $q_1 = n(i)$ and $a_{q_j}<a_i$ for all $1\leq j<k$ and if s_q is a B-segment with $q_j<q<q_{j+1}$ for some $1\leq j<k$ then $a_q\leq a_{q_j}$.

B(i) is always defined unless s_i happens to be the last segment in the double comb. B(i) consists of exactly those segments s_i would see if there were no T-segments to block s_i's vision. Hence B(i) can be determined for all B-segments by applying the rooted-case algorithm to the list of B-segments while ignoring the T-segments.

T(i) = $\{s_{p_j}\}_{j=1}^{m}$ where each s_{p_j} is a T-segment and s_{p_1} is the first T-segment with $i<p_1$ and [illegible] $p_j<p_{j+1}$, $b_{p_j}>b_{p_{j+1}}$ and if s_p is a T-segment such that $p_j<p<p_{j+1}$ for some j then $b_p\geq b_{p_j}$.

T(i) may be empty if there are no T-segments to the right of s_i whose projections onto the x-axis overlap that of s_i's. T(i) consists of exactly those T-segments which would be visible to s_i on its right, if there were no other B-segments to block s_i's vision.

Clearly RV(i) ⊆ B(i) ∪ T(i). RV(i) can be determined by scanning both B(i) and T(i) in ascending x order (from left to right), and halting when either of the two sequences interfere with s_i's vision of the other. The definition below gives the first pair of segments, one from each sequence, that blocks s_i's vision on its right.

Define (g,h) such that $1\leq g\leq k$, $1\leq h\leq m$ and $a_{q_g}\geq b_{p_h}$ and if $p_h<q_g$ then $b_{p_h}>a_{q_{g-1}}$

else (if $q_g<p_h$ then) $b_{p_{h-1}}>a_{q_g}$.

[If $q_g < p_h$ and $a_i \leq a_{q_g}$ then let $h = 0$. If $b_{p_h} = 0$ and $p_h < q_g$ then let $g = 0$.]

If such a pair (g,h) exists, then

$$RV(i) = \{s_{q_k}\}_{k=1}^{g} \bigcup \{s_{p_k}\}_{k=1}^{h}.$$

If such a pair does not exist then B(i) and T(i) do not interfere with each other and RV(i) = B(i)∪T(i). So in this case define (g,h) = (k,m). It is clear that (g,h) can be found by scanning T(i) and B(i) simultaneously and that the number of segments examined is no more than the number of visibility pairs reported within a constant. Exactly the same concept applies for computing RV(i) for T-segment i. We shall give below a brief description of the algorithm for computing RV(i) for each B- and T-segment. Note that we should not compute B(i) and T(i) explicitly for each B- or T-segment, otherwise the time taken may be more than what we want because RV(i) may contain much fewer elements than those contained in B(i) and in T(i). As we shall see, the time needed for computing RV(i) for all i is proportional to the number of visibility pairs reported plus the total number of B- and T-segments examined.

We shall scan the lists of T- and B-segments in sorted x-order from right to left. During the scan we shall keep a list of T-segments $t_1, t_2, \ldots, t_k$ that we have scanned in that order with $b_{t_1} < \ldots < b_{t_k}$ and a list of B-segments $j_1, j_2, \ldots, j_m$ with $a_{j_1} > a_{j_2} > \ldots > a_{j_m}$. When a T-segment t is seen, the list T is scanned backward and i_q is deleted from T and included in RV(t) if [illegible], and the list B is scanned forward from j_1 and j_q is deleted from B and included in RV(t) if $a_{j_q} \geq b_t$. The segment t is then appended to the list T. When a B-segment s is seen, we do the same except that the segment is appended to the list B. (For an example see Fig. [illegible])

4. General case

In this section we consider the general problem and propose a method to divide it into subproblems which are double combs. Then solve each double comb case separately and combine the results together. The complete algorithm takes time O(N), where N is the total number of segments after the division.

Endwhile

If N is determined to be small enough, then the problem can be easily divided into double comb subproblems, using the UG(i)'s and LG(i)'s. The subproblems will be numbered from one to Count. Each subproblem must be provided with its T and B lists of segments in sorted x-order. These lists are built simultaneously in one scan of the entire set of segments, in x-order. As each segment s_i is encountered, it is appended to the B-list of UG(i), the T-list of LG(i) and the B-list of any number properly between UG(i) and LG(i).

Either the algorithm with or without sorting may be used to solve the subproblems. However one more scan is necessary to provide sortings by a_i and b_j. Care must be taken to satisfy the assumptions on the a_i's and b_j's, which are made by the double comb algorithm. These can be satisfied by using a bucket sort to order all endpoints of equal y-coordinates by [illegible] x-coordinate, and by considering all b_j's of a given value before any a_i's. The [illegible] N [illegible] will be replaced by the values of the horizontal lines bounding the given subproblem.

Once each subproblem has been solved, some processing may be required to remove duplicate [illegible]. This can be accomplished in linear time in the number of segments, n, using a bucket sort.

There are certain types of problems in which N is guaranteed to be linear in n.

Lemma 7 If all segments are of equal length then $N \leq 2n$.

Proof. It will be shown that UG(i)=LG(i)-1 for all s_i. Suppose not. Then there must be some segment s_j which was inserted in LG(i)-1 and deleted in LG(i). Since s_i was inserted in UG(i)<LG(i)-1, then s_i must be longer than s_j, but this is not possible.

□

Similarly, if the maximum ratio of lengths is C, then N is at most (C+1)n since each segment can be cut up into at most C+1 pieces.

If the amount of containment is also bounded by C, then N is at most (C+1)n. The number of other segments whose projections onto the y-axis are disjoint but all contained in that of a segment, s_i, is defined to be the containment number of s_i. Clearly, s_i will not be divided into more pieces than its containment number plus one.

There are unfortunately, examples in which N can be $O(n^2)$, as illustrated in the example of Fig. 8, where there are $n/2$ long segments and $n/2$ short segments. Each short segment will be divided into two while each long one into $(n/2+1)$. The total is $n+(n/2+1)n/2 = n^2/4 + 3n/2$.

The dividing algorithm presented at the beginning of this section which takes O(n) time can be used to find out what the exact value of N is for any specific input before actually running the whole algorithm. If N turns out to be large, say, larger than n log n, then one may want to use the following O(n log n) time algorithm instead.

If only an $O(n \log^2 n)$ algorithm is desired, then the algorithm becomes quite simple since log n time may be spent to insert a segment into a list. The sorting of the a_i's and b_i's is not essential since a scan by x_i can be used, however the simple algorithm presented here uses these sortings as well as the x-sorting.

The segments will be scanned in decreasing y-coordinate, a list of all segments in x-order crossing a horizontal scan line will be kept. In order to insert a new segment into the list, a tree will be used. Each node will have an empty flag, and each edge will also have an empty flag which will be set if all nodes in the subtree below it, are empty.

Step 1 Build a balanced binary tree, using the n segments (ordered by x_i's, as nodes.

Step 2 Consider each y which is either a b_i or an a_i.

Step 2.1 For each $a_i = y$

Find a neighbor in the list and insert, by starting at i in the tree and using the empty flags to find a neighbor. Reset i's empty flag, and the empty flags of the edges between i and the root.

Step 2.2 For each $a_i = y$, report visibility pairs with its neighbors in the list.

Step 2.3 For each $b_i = y$, report visibility pairs of its neighbors, if neither neighbor has $b_j = y$.

Delete i from the list. Set i's empty flag and if both edges from i (below) are empty, travel from i to the root, setting empty flags until either a non-empty node or an adjacent non-empty edge is encountered.

Repeat until all segments have been inserted and deleted.

5 Conclusions

In this paper, we studied a visibility problem arising from VLSI layout compaction. Various complexity results have been obtained. Assuming the input being sorted in their x- and y- coordinates, we proposed linear time algorithms for various special cases and presented a dividing algorithm for the general problem, whose time complexity was linear in the number of final segments.

However, it remains open whether the general visibility problem can be solved in linear time.

Appendix 1. Proof of Lemma 5

Several problems related to the union of k-dimensional intervals have been raised in computational geometry. The original formulations can be found in [8],[9]. In particular, for k = 1, Klee posed the question whether the measure of $\bigcup_{i=1}^{n} [a_i,b_i]$ can be found in less than O(n log n) steps [9]. A negative answer to Klee's question was provided by Fredman and Weide in [10], where they showed that the measure problem requires Ω (n log n) steps under the Decision Tree computational model with Linear Comparisons (DTLC). (In DTLC, each node of a decision tree is labeled with a linear function of the inputs, which is computed upon traversal of the node. The value v of this function indicates whether to proceed to the left son (v > 0) or to the right son (v ≤ 0), or v is taken as the answer when computed at a leaf. In a decision problem, each leaf is labeled with "yes" or "no" instead of a function.)

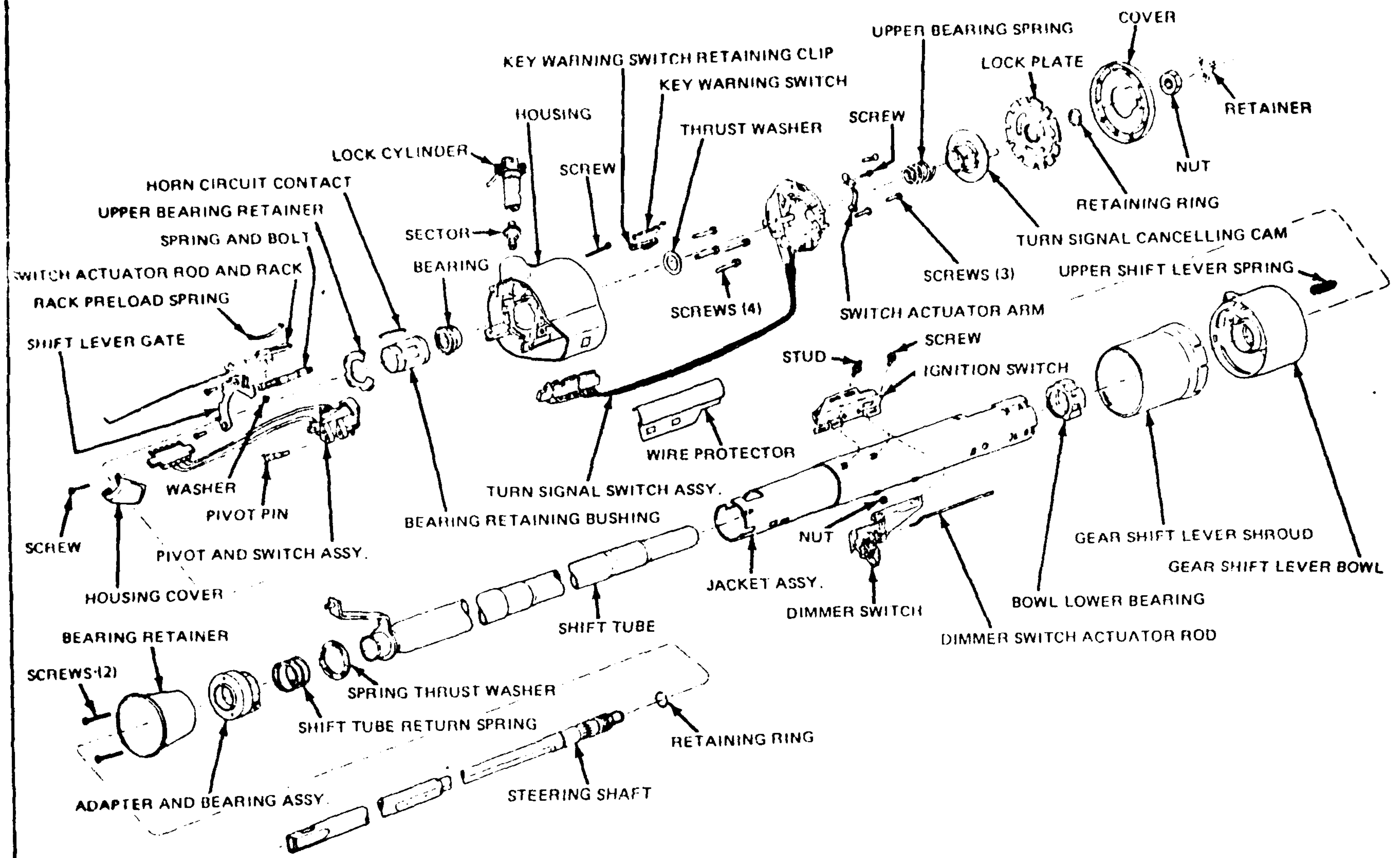
In this appendix we consider the COMPACTNESS decision problem. This is the problem of determining whether $\bigcup_{i=1}^{n} [a_i,b_i]$ is an interval, or if it splits into several disjoint intervals. COMPACTNESS can be trivially solved in O(n log n) steps, by sorting the intervals in the a_i-order, and then building the interval union by adding one interval at a time. However, we prove here that this problem actually requires Ω (n log n) steps. Our argument is similar to that used in proof 2 of the main theorem of [10].

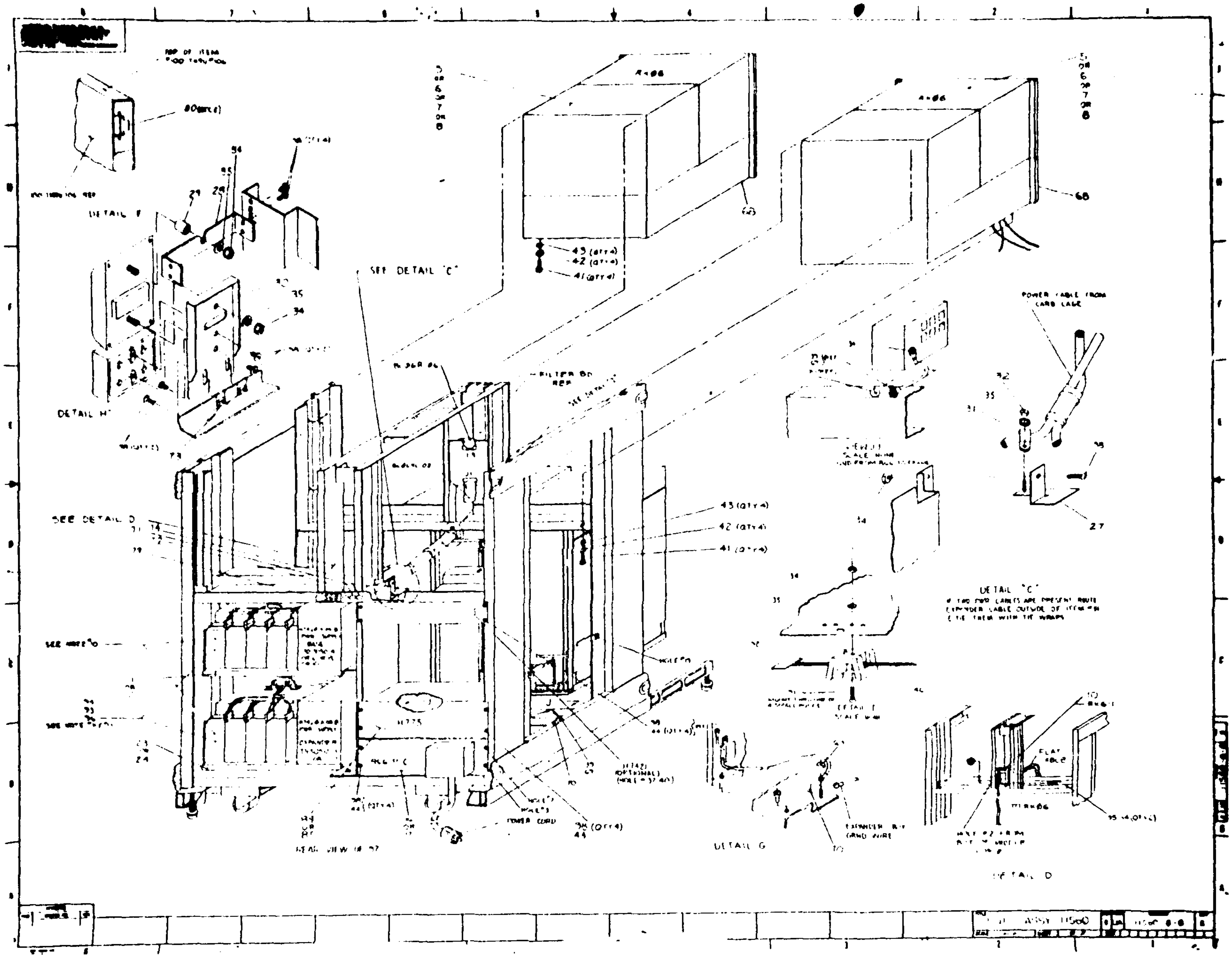
Claim. COMPACTNESS requires Ω (n log n) steps in DTLC model.

Proof. Let the endpoints a_i,b_i of the n input intervals be treated as coordinates in a 2n-dimensional space, thus establishing a one-to-one correspondence between ordered sets of n intervals and points in such a space. We will use the terms sets of intervals and points interchangeably. If two points P′, P″ trace the same path γ in a DTLC algorithm for COMPACTNESS, they lead to the same leaf, and receive the same answer "yes" or "no". In fact, the linear inequalities encountered along γ define a convex region Γ in the space, to which P′ and P″ belong together with all the points of the straight line segment $\overline{p'p''}$. All the points of Γ will receive the same answer from the algorithm.

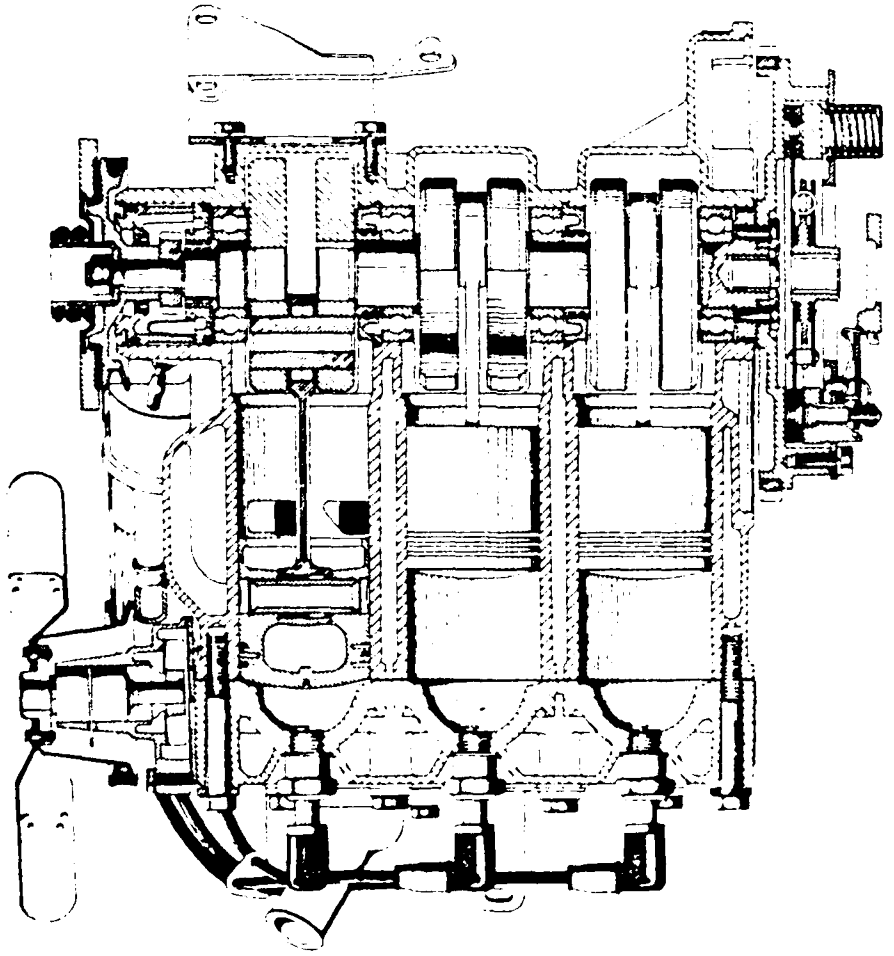
Consider now the n intervals $I_1 = [0,1], I_2 = [1,2], \ldots, I_n = [n-1,n]$. There are (n-1)! permutations $\pi_1, \pi_2, \ldots, \pi_{(n-1)!}$ of such intervals, such that $\pi_i(1) = I_1$ for all i. (That is, the first interval is always in the first position). Each of the above permutations π_i corresponds to

STANDARD COLUMN





DETAIL F
DETAIL H
SEE DETAIL "C"
SEE DETAIL D
A×86
68
43 (QTY 4)
42 (QTY 4)
41 (QTY 4)
80 (QTY 2)
FILTER BD REF
POWER CABLE FROM CARD CAGE
DETAIL "C"
DETAIL E
H775
POWER CORD
98 (QTY 4)
44
REAR VIEW OF 97
DETAIL G
EXPANDER BX GRND WIRE
FLAT CABLE
DETAIL D
27
ASSY 11/60



Representation, manipulation, and reasoning about physical objects

John Hopcroft

Cornell University
Ithaca, New York

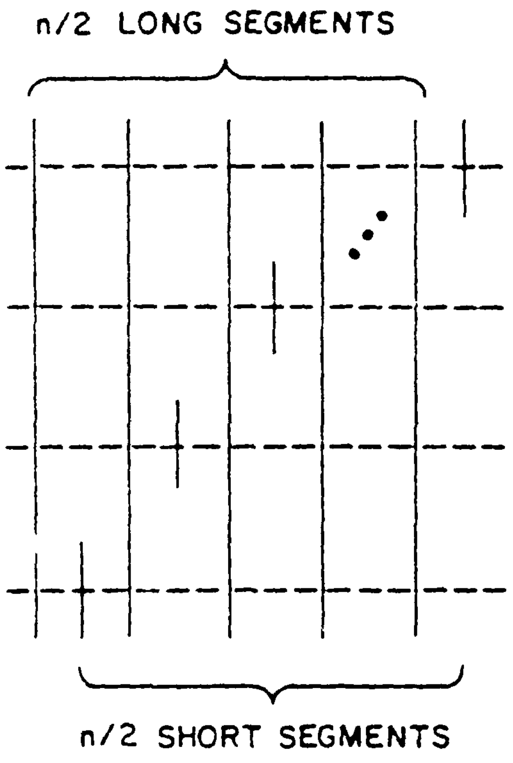
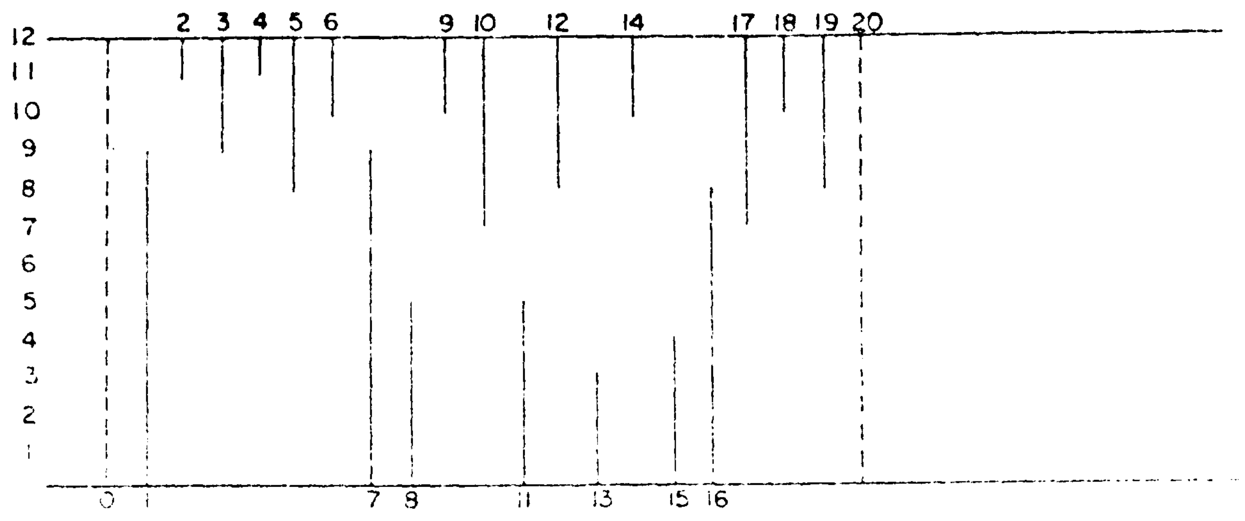


Fig. 8. Number of final segments is $O(n^2)$

EXAMPLE: DOUBLE COMB CASE (b): WITHOUT SORTING



	GENERATED	
$B(1) = \{7\}$	$T(1) = \{3,5\}$	$RV(1) = \{3,5,7\}$
$B(7) = \{8,16\}$	$T(7) = \{10\}$	$RV(7) = \{8,10,16\}$
$B(8) = \{11\}$	$T(8) = \Phi$	$RV(8) = \{11\}$
$B(11) = \{13,15,16\}$	$T(11) = \Phi$	$RV(11) = \{13,15,16\}$
$B(13) = \{15\}$	$T(13) = \Phi$	$RV(13) = \{15\}$
$B(15) = \{16\}$	$T(15) = \Phi$	$RV(15) = \{16\}$
$B(16) = \Phi$	$T(16) = \{17\}$	$RV(16) = \{17\}$

Fig. 7 (Part 3) Double comb case (without sorting)

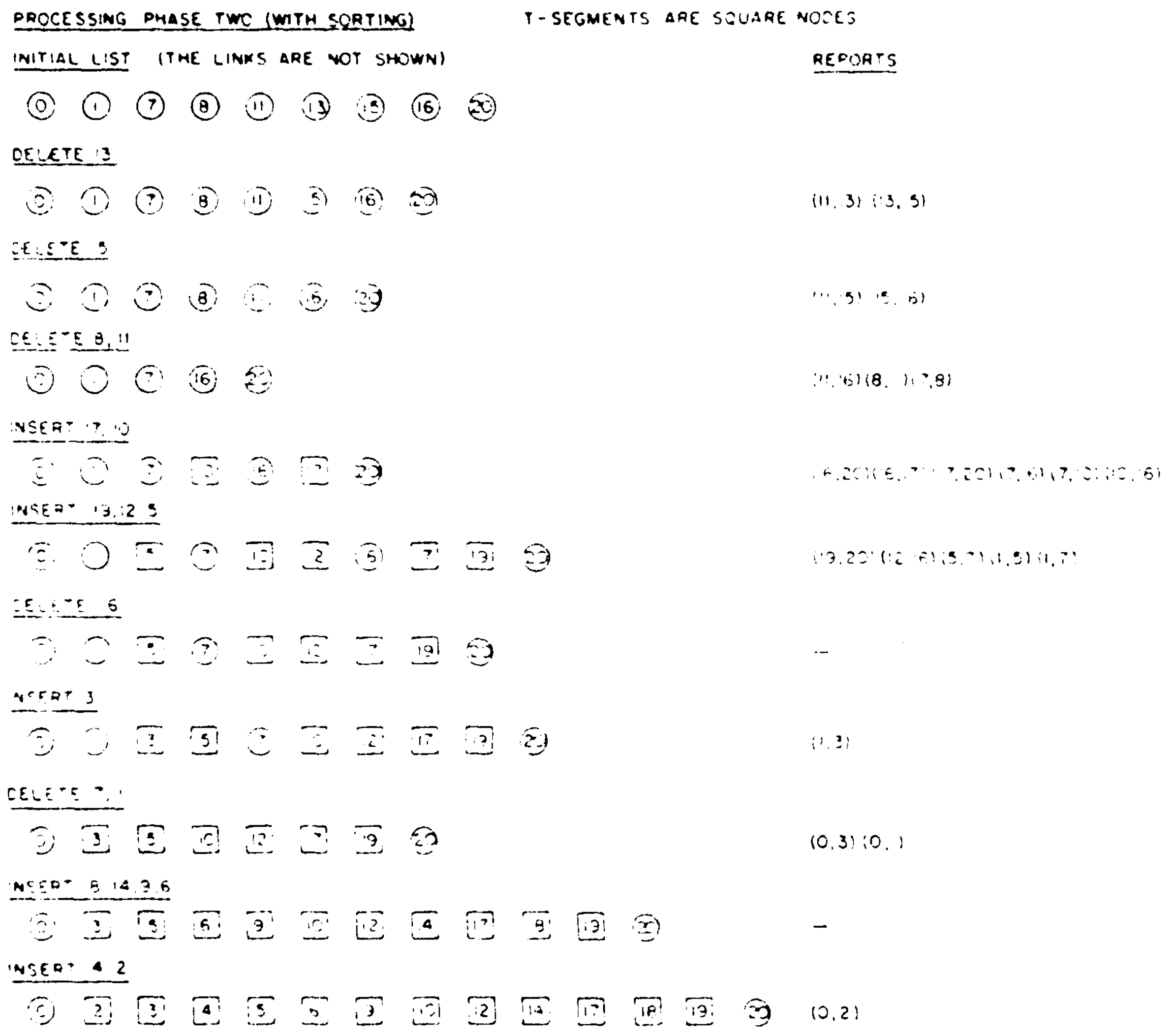
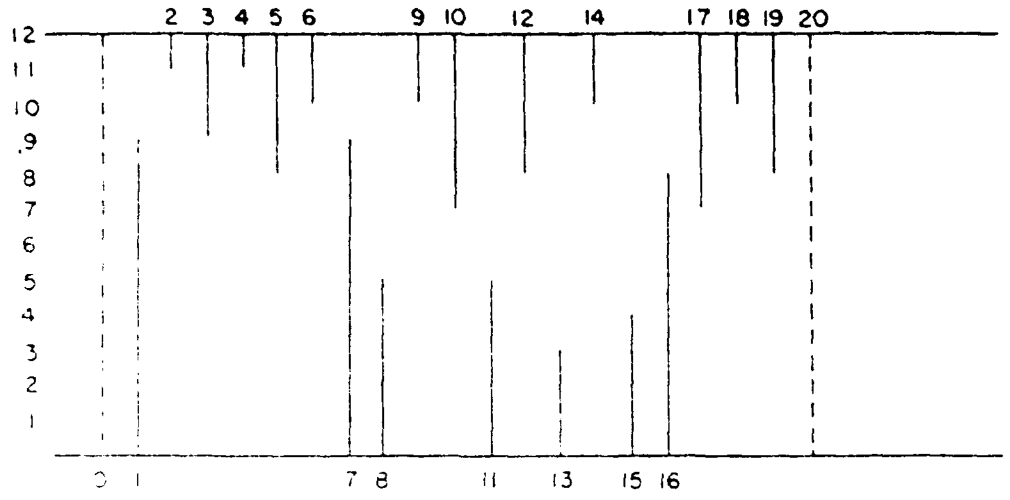


Fig. 7 (Part 2). Double comb case (with sorting)

EXAMPLE DOUBLE COMB CASE (a) WITH SORTING



RESULT OF PHASE ONE

PAIRS REPORTED (2,3)(3,4)(4,5) (5,6) (6,9) (9,10) (10,12) (12,14) (14,17) (17,18) (18,19) (3,5) (5,10) (12,17), (17,19), (17,20)

VTL VTR INFORMATION

T-SEGMENT	VTL	VTR	AP's OBTAINED DURING PHASE TWO & UPDATES
2	0	3	3
3	0	5	5
4	3	5	5
5	0	0	7
6	5	9	9
9	6	10	10
10	0	0	16
12	10	0	16
14	12	17	17
17	0	20	20
18	17	19	19
19	17	20	20

Fig. 7 (Part 1) Double comb case (with sorting)

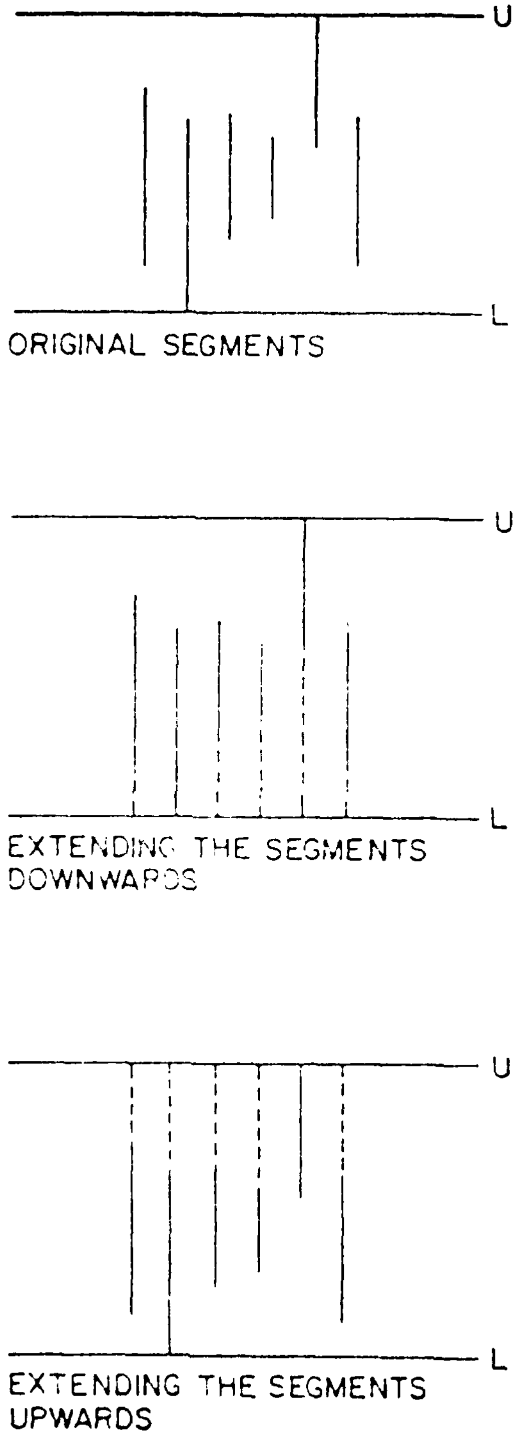

U
L
ORIGINAL SEGMENTS
U
L
EXTENDING THE SEGMENTS DOWNWARDS
U
L
EXTENDING THE SEGMENTS UPWARDS

Fig. 6 The overlapping case

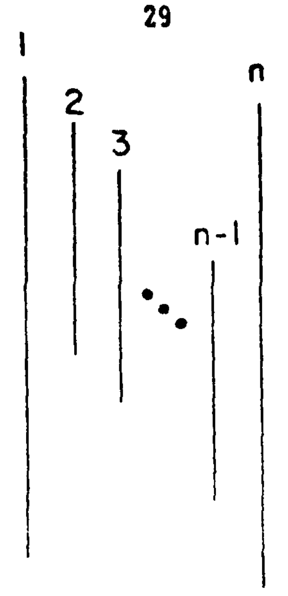
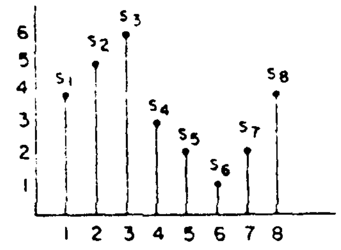


Fig. 4. An example to illustrate Lemma 3



TIME THROUGH THE OUTER WHILE LOOP	LIST BEFORE EXECUTION	C BEFORE EXECUTION	SEGMENT(S) DELETED	VISIBILITIES REPORTED
1	0 1 2 3 4 5 6 7 8 9	8	s_1	(0,1), (1,2)
2	0 2 3 4 5 6 7 8 9	7	s_2	(0,2), (2,3)
3	0 3 4 5 6 7 8 9	6	s_6	(5,6), (6,7)
4	0 3 4 5 7 8 9	5	s_5, s_7	(7,8), (5,7), (4,5)
5	0 3 4 8 9	3	s_4	(3,4), (4,8)
6	0 3 8 9	2	s_8	(3,8), (8,9)
7	0 3 9	1	s_3	(0,3), (3,9)
	0 9	STOP ⟶ 0		

VISIBILITY LISTS

V(0) = 1,2,3	V(5) = 6,7,4
V(1) = 0,2	V(6) = 5,7
V(2) = 1,0,3	V(7) = 6,8,5
V(3) = 2,4,8,0,9	V(8) = 7,4,3,9
V(4) = 5,3,8	V(9) = 8,3

Fig. 5. Example of the rooted case

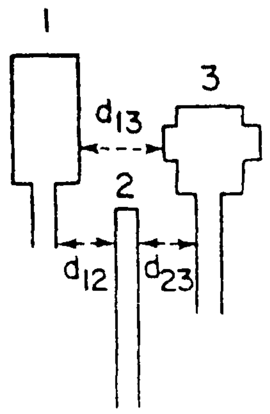


Fig. 2 Distance between elements

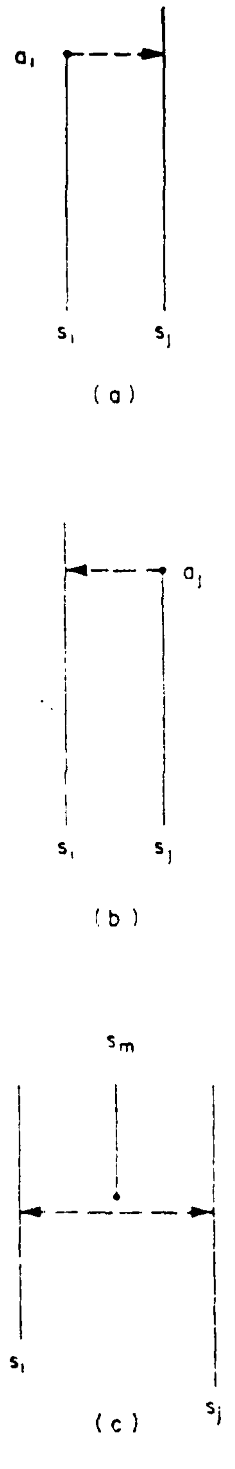
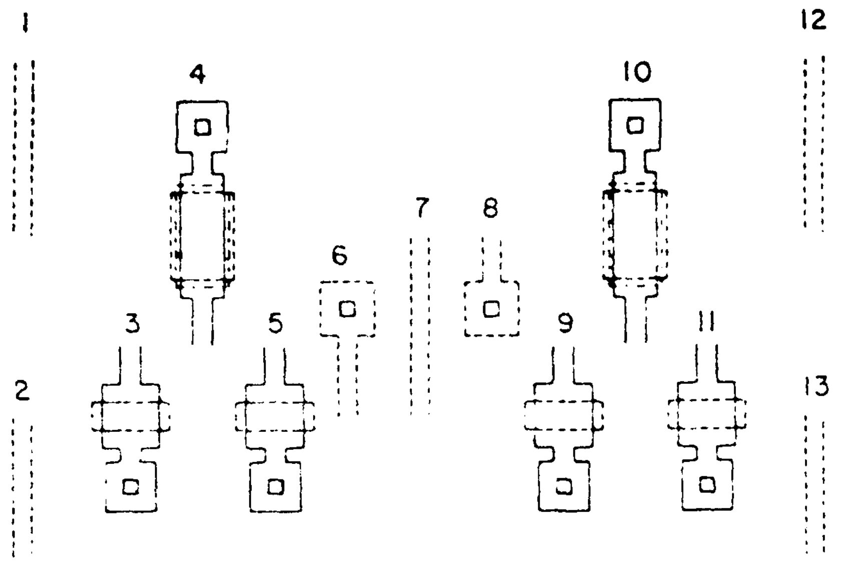
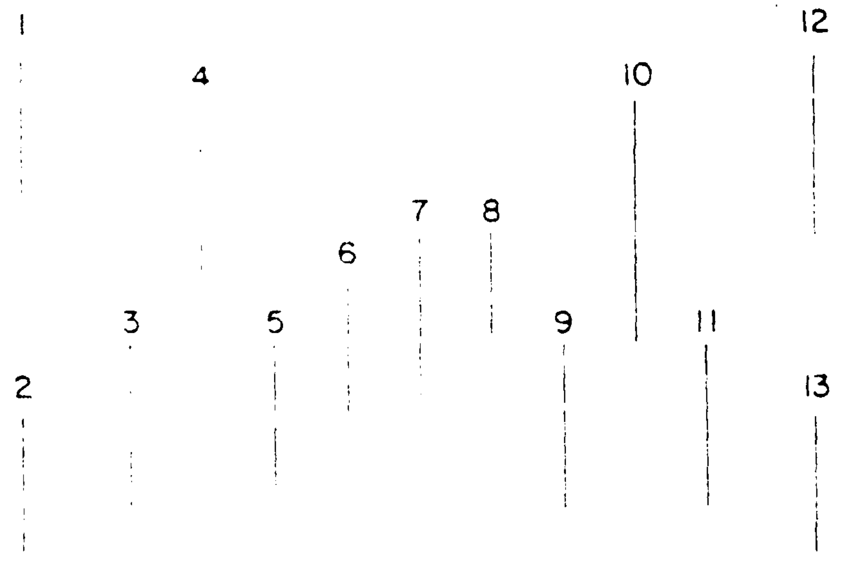


Fig. 3. Illustration for the proof of Lemma 2



(a)



(b)

Fig. 1 Original shapes and their simplified representations

References

[1] C. Mead and L. Conway, Introduction to VLSI Systems, Addison-Wesley, 1980

[2] M. Y. Hsueh, "Symbolic Layout and Compaction of Integrated Circuits," ERL Memo NO. UCB ERL M79 80, Univ. of California, Berkeley, Dec. 1979

[3] A. E. Dunlop, "SLIM-The Translation of Symbolic Layout into Mask Data," Proc. 17th Design Automation Conf., June 1980, pp. 595-602.

[4] R. Auerbach B. Lin, and E. Elsayed, "Layout Aid for the Design of VLSI Circuits," Computer Aided Design, Vol. 13, No. 5, pp. 271-276, Sept. 1981.

[5] N. Weste, "Virtual Grid Symbolic Layout," Proc. 18th Design Automation Conf., June 1981, pp. 225-233

[6] K. H. Keller, A. R. Newton, and S. Ellis, "A Symbolic Design System for Integrated Circuits," Proc. 19th Design Automation Conf., June 1982, pp. 460-466.

[7] Y. A. Liao and C. K. Wong, "An Algorithm to Compact a VLSI Symbolic Layout with Mixed Constraint," IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems (CAD ICAS), to appear.

[8] F. Lodi, F. Luccio, C. Mugnai, L. Pagli, "A Preliminary Investigation on Two Dimensional Data Organization," Proc. Annual Conf. AICA, 1976.

[9] V. Klee, "Can the Measure of $\cup [a_i,b_i]$ be Computed in Less Than O(n log n) Steps",' Amer. Math Monthly 84, 4 (April 1977), 284-285.

[10] M. L. Fredman and B. Weide, "On the Complexity of Computing the Measure of $\cup$ $[a_i,b_i]$, Comm. ACM 21, 7 (July 1978), 540-544.

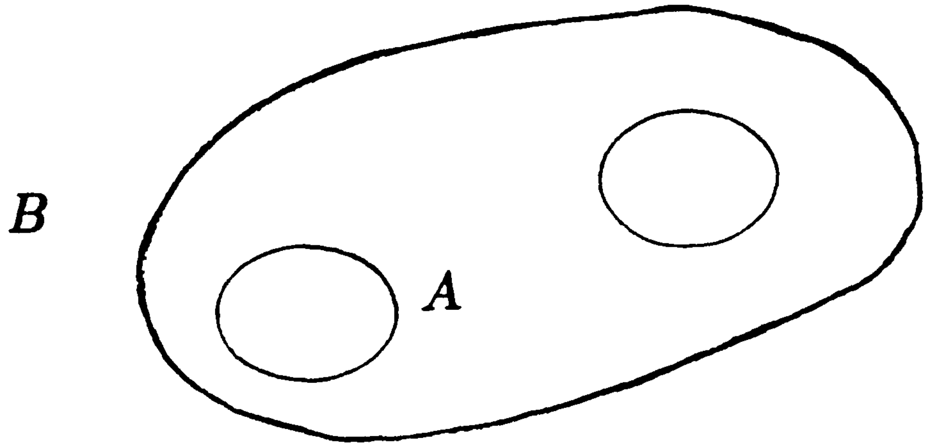
[11] C. L. Liu, Introduction to Combinatorial Mathematics, McGraw-Hill, Inc., New York, 1968.

an ordered set of intervals, hence to a point P_i, and all these points receive answer "yes" in DTLC, because $\cup I_i$ is clearly an interval. We prove now that no pair P_i, P_j of such points can trace the same path in DTLC. In fact, if this were the case, each point P on $\overline{P_iP_j}$ would receive answer "yes". It is straightforward to note that any such point P corresponds to an ordered set of intervals of unit length. However, if $r>1$ is the smallest index value for which $\pi_i(k) = I_r, \pi_j(k) = I_s, r<s$, for some k, as soon as a point P starts sliding from P_i to P_j, the union of the intervals of P breaks up into two intervals, around the coordinate value r-1. Hence, there is at least one point P on $\overline{P_iP_j}$ whose answer must be "no", against the hypothesis, that is, P_i and P_j must correspond to different leaves.

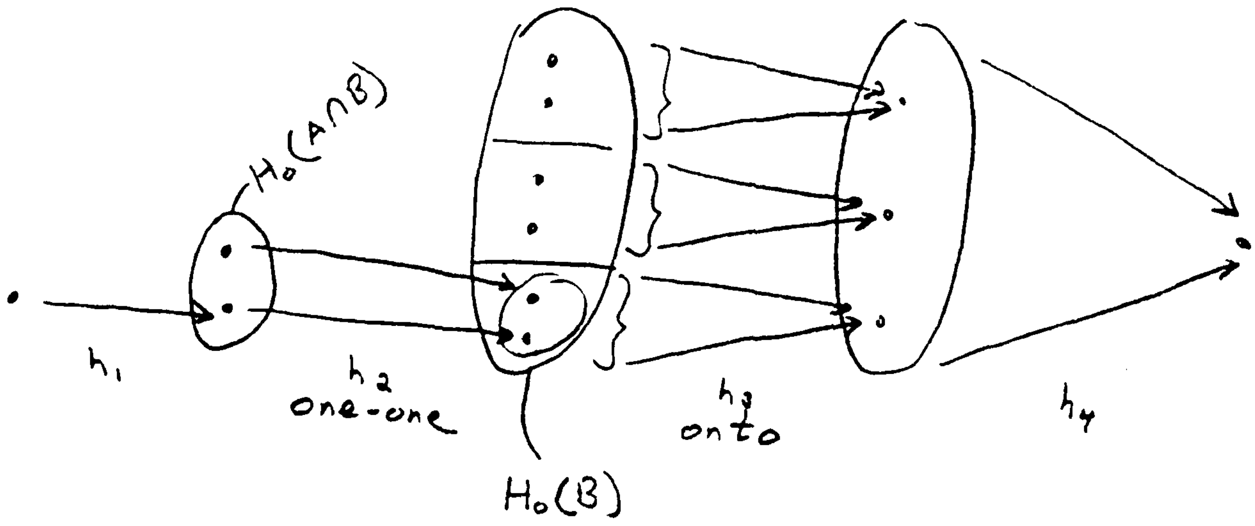
We conclude that DTLC must have at least (n-1) distinct leaves, that is, its depth is Ω (n log n)

□

The above proof makes use of segments of equal length. Therefore, it also proves that COMPACTNESS remains of Ω (n log n) in the restricted case $a_i - b_i = c$, $1 \leq i \leq n$ (i.e., all intervals have the same length).



$$\underbrace{H_1(A \cup B)}_{\emptyset} \xrightarrow{h_1} \underbrace{H_0(A \cap B)}_{\text{boundary}} \xrightarrow{h_2} \underbrace{H_0(A)}_{Z} \oplus \underbrace{H_0(B)}_{\substack{\text{valid}\\ \text{configurations}}} \xrightarrow{h_3} \underbrace{H_0(A \cup B)}_{Z} \xrightarrow{h_4} \emptyset$$



Stereophenomenology

stereo - solid

phenomenology
- theory of representation

Representing, manipulating and reasoning about physical objects electronically.

What does science include?

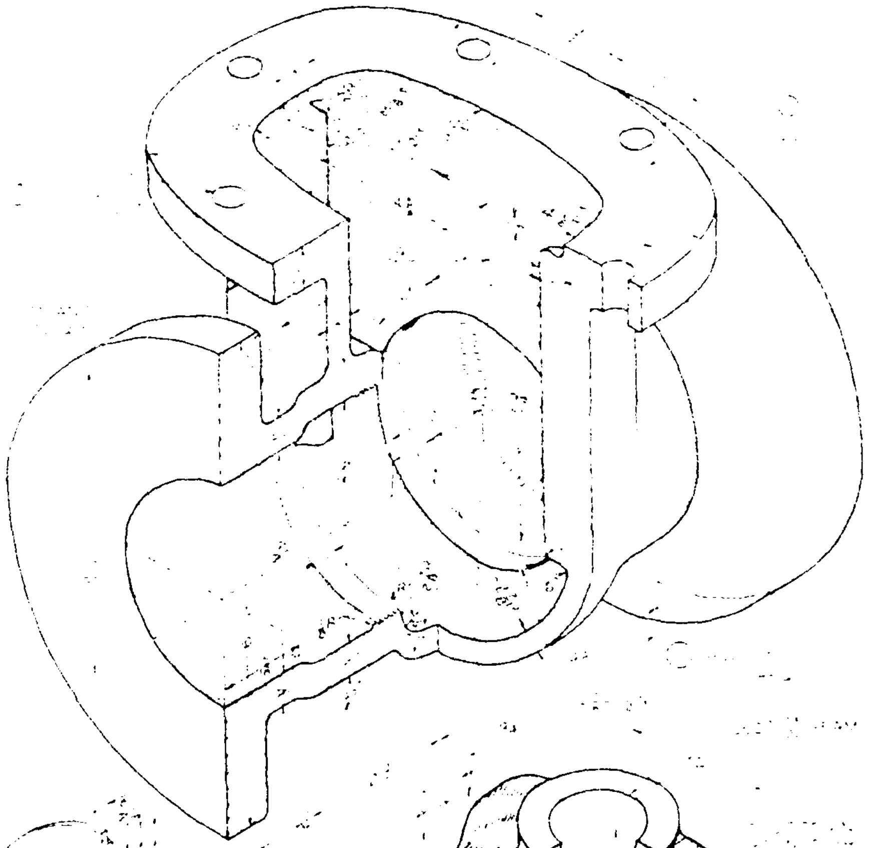
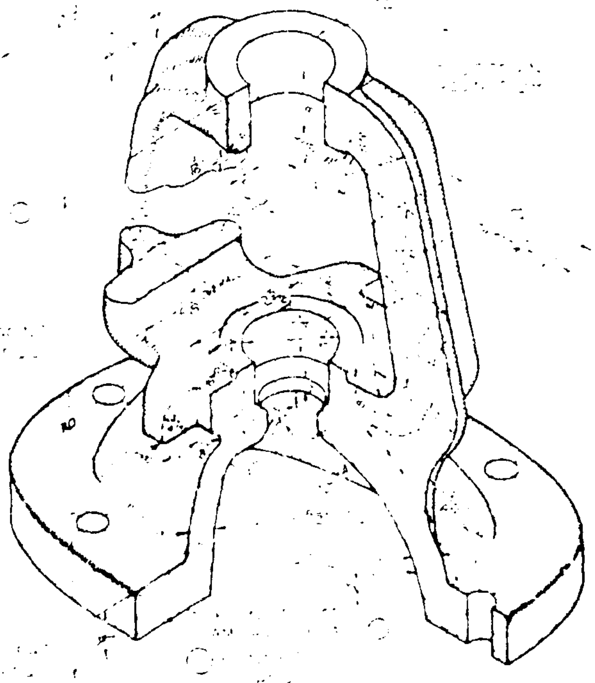
representation of objects
- surfaces and solids
- functional dependency
- hierarchical view
- abstract models - features, etc
- generic objects
- internal structure
- flexible and nonrigid objects

algorithms
- display
- intersection
- motion planning

user interfaces
- editing
- interactive graphics
- attribute grammars - simplifying local changes

reasoning about objects
- grip positions
- external forces
- shape
- design for function

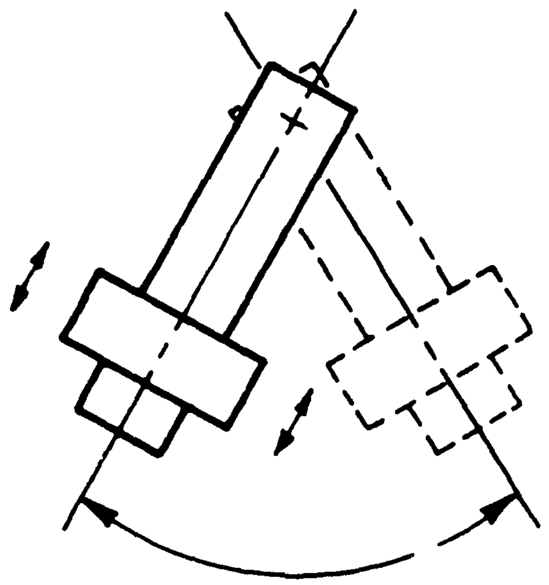
manipulation
- gripping strategies
- object motion

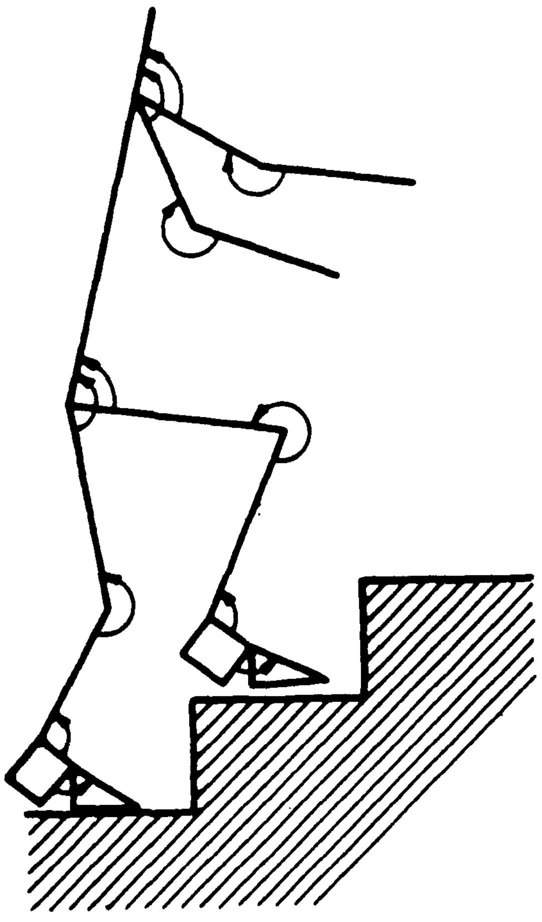



```
pipe(radius, thickness, length) := (cyl1 - cyl2) ∩ H1 ∩ H2  where
   begin
      cyl1 := ycylinder(radius+thickness);
      cyl2 := ycylinder(radius);
      H1 := {y ≥ 0};
      H2 := {y ≤ length};
      top := (cyl1 - cyl2) ∩ {y = length};
      bottom := (cyl1 - cyl2) ∩ {y = 0};
      outside := cyl1.surface ∩ H1 ∩ H2;
      top.in_edge := top ∩ cyl2;
      bottom.in_edge := bottom ∩ cyl2
   end;

flanged_pipe(radius, thickness, length) :=
      smooth_attach(nipple.top.in_edge, sflange.bottom.in_edge, 1/8)  wh
   begin
      ft := 5*thickness+hole_diameter;
      flange := pipe(radius, ft, f);
      sflange := smooth(flange.outside, flange.bottom, 1/16);
      nipple := pipe(radius, thickness, length);
      bottom := nipple.bottom;
      top := sflange.top
   end;

ovoid(r1, r2, r3) := smooth(cyl1 ∩ cyl2 ∩ cyl3, r3)  where
   begin
      cyl1 := ymove(-(4+9/16), zcylinder(r2));
      cyl2 := ymove(4+9/16, zcylinder(r2));
      cyl3 := zcylinder(r1);
   end;
```

```
x-slice(width) := H1∩H2 where
   begin
      H1:={x ≥ 0};
      H2:={x ≤ width};
      left:={x = 0};
      right:={x = width}
   end;

y-slice(length) := H1∩H2 where
   begin
      H1:={y≥0};
      H2:={y≤length};
      front:=H2;
      back:=H1
   end;

z-slice(height) := H1∩H2 where
   begin
      H1:={z ≥ 0};
      H2:={z ≤ height};
      top:=H1;
      bottom:=H2
   end;

cuboid(length, height, width) :=
      x-slice(width) ∩ y-slice(length) ∩ z-slice(height) where
   begin
      front.right-edge := front ∩ top;
      front.right.top-vertex := front ∩ right ∩ top
   end;
```

```
vertex1:=(x-, y-, z-coordinate);
    ...
vertex8:=(x-, y-, z-coordinate);
edge1:=line(vertex1, vertex2);
    ...
edge12:=line(vertex7, vertex8);
front-face:=patch(edge1, edge2, edge3, edge4);
right-face:=patch(edge4, edge8, edge10, edge11);
    ...
```





Comparison of techniques

abstract models versus solid models

solid models versus surface representations

polygonal patches versus bicubic patches
versus algebraic surfaces

numerical techniques versus symbolic

More general objects

nonrigid
- plastic flowing into mold
- sail

shape determined by external forces
- coil spring

generic or parameterized objects

Design for functionality

can objects be represented by function rather than shape and dimension

Theorem (simplest form). If there exists a motion of two objects from an initial position where they are in contact to a final position where they are in contact then there exists a motion whereby the objects remain in contact at all times.

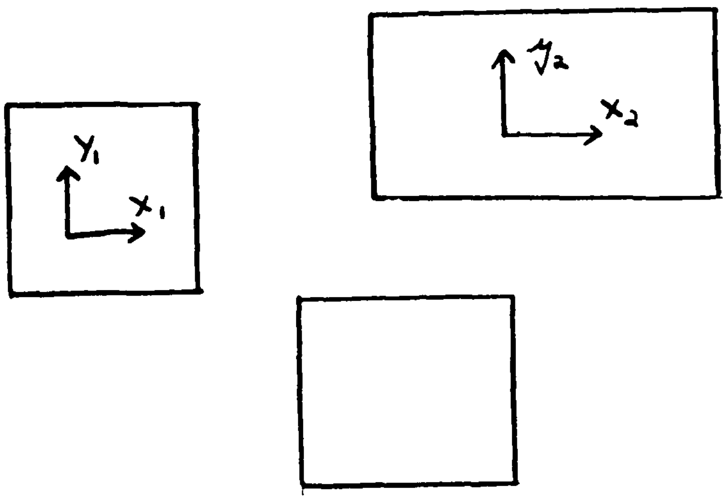
Motion is continuous but point of contact is not.

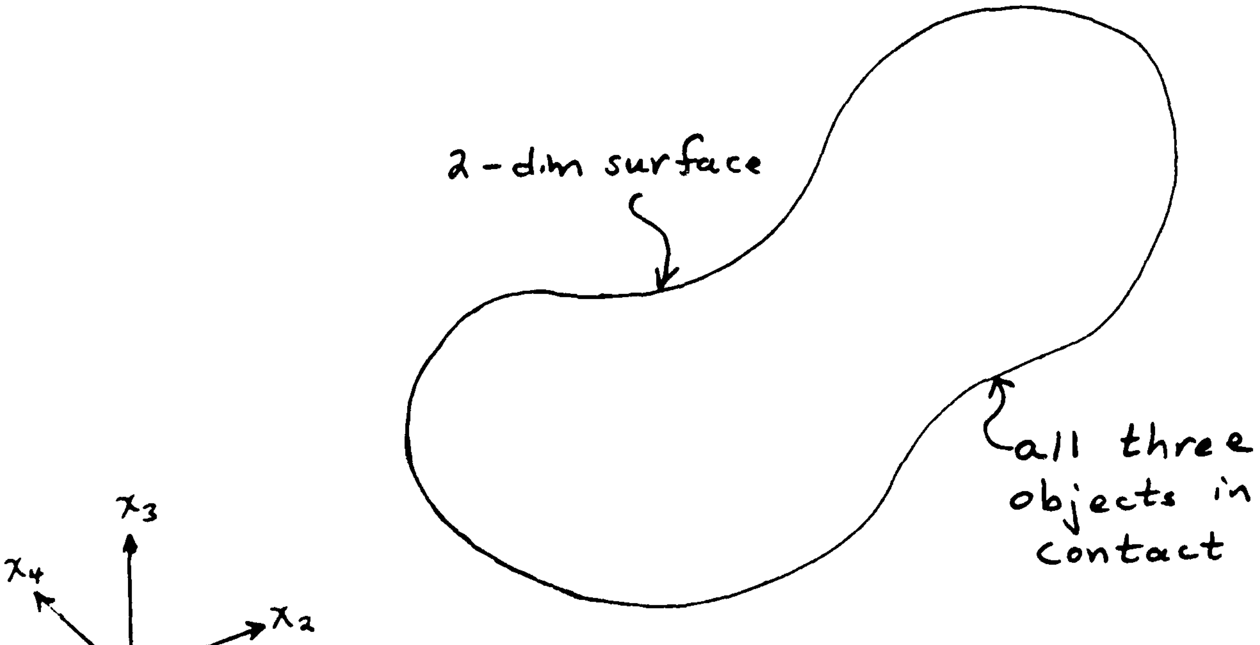
Motivation

1. Trying to place problems in PSPACE-linkage motion, block motion

2. Simple multiple object motion planning

3. Compliant motion

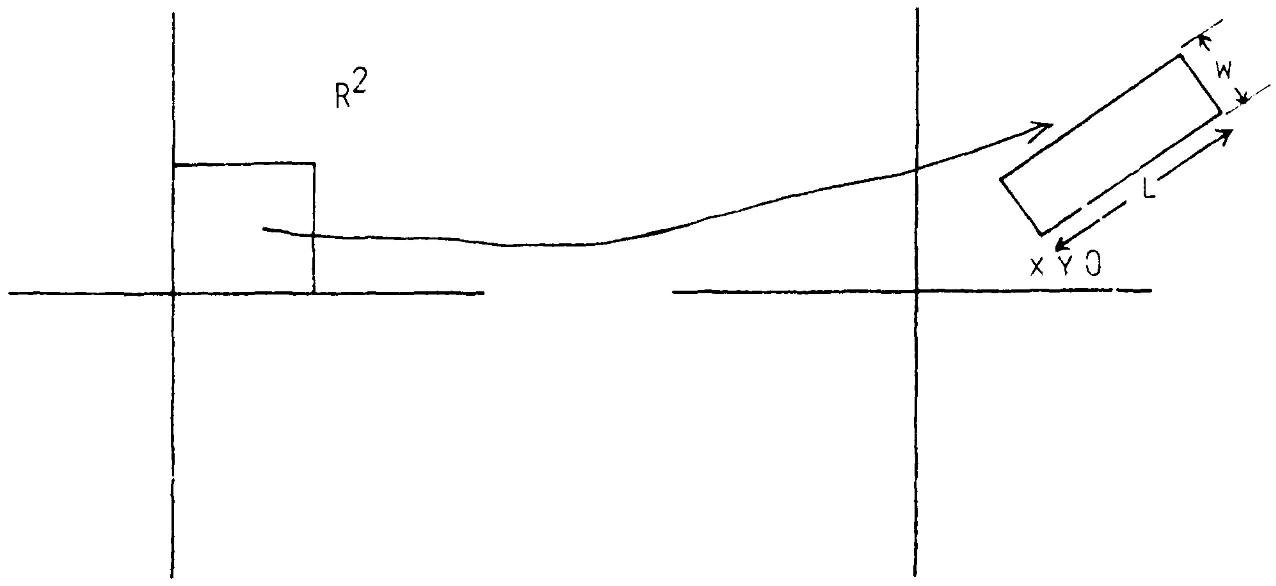

y_1
x_1
y_2
x_2


2-dim surface
all three objects in contact
x_3
x_4
x_2
x_1

4-dim space

HOW DO WE REPRESENT GENERIC OBJECTS? POSSIBLY BY REPRESENTING ABSTRACT OBJECTS WHOSE POSITION, SIZE AND SHAPE ARE INSTANTIATED ONLY WHEN SPECIFIC INSTANCE NEEDED.

ALLOWS MOTION PLANNING AND OTHER ALGORITHMIC TASKS TO BE CARRIED OUT FOR CLASSES OF OBJECTS RATHER THAN INDIVIDUAL OBJECTS.

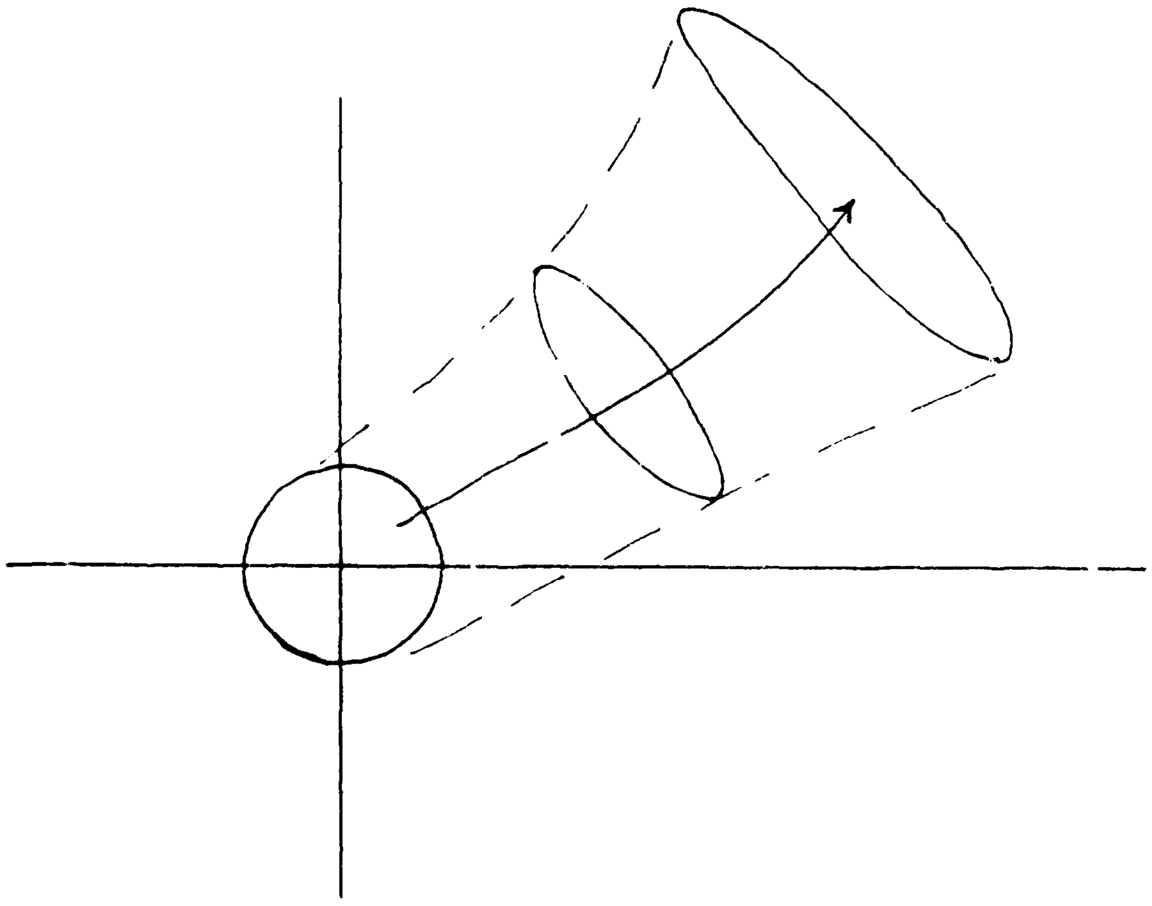


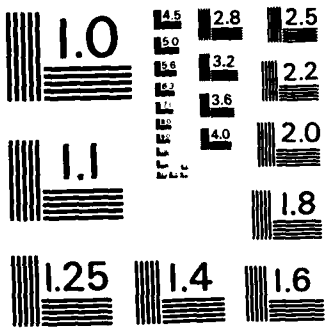
AN OBJECT IS A PARAMETERIZED MAP FROM A CANONICAL REGION OF R^3 TO R^3

MOTION IS A CONTINUOUS MAPPING FROM [0,1] TO PARAMETER SPACE.

EXAMPLES

- TRANSLATION
- ROTATION
- GROWTH
- CONTINUOUS DEFORMATION




MICROCOPY RESOLUTION TEST CHART

NATIONAL BUREAU OF STANDARDS - 1963 - A

Lemma: Configuration space is path connected.

Lemma: $H_0(S) = Z^k$ if k path connected components

Lemma: $H_1(S) = \phi$ if space contractible to a point

Mayer-Vetoris Theorem:

$$H_1(A \cup B) \xrightarrow{h_1} H_0(A \cap B) \xrightarrow{h_2} H_0(A) \oplus H_0(B)$$

$$\xrightarrow{h_3} H_0(A \cup B) \xrightarrow{h_4} \emptyset$$

is an exact sequence.

exact sequence image of h_i = kernel of h_{i+1}

conditions for theorem to hold

1. Translation is needed in order that configuration space object be path connected.

   Alternative Lemma: There exists two paths, one in free space, one in the configuration space object.

2. Space must be contractible to a point.